INFORMATIQUE ET LINGUISTIQUE
INTELLIGENCE ARTIFICIELLE
MUSIQUE ET INFORMATIQUE

1973 - 1974
UNIVERSITÉ PARIS VIII
1974 - 1975

INITIATION AU LANGAGE - MACHINE EN QUASI-11

P. GREUSSAY

Nous programmeurs aimons voyager d'une machine à l'autre. D'abord parceque les paysages qui changent font du bien à l'esprit. Mais je crois aussi que, plus ou moins consciemment, nous pressentons que programmer trop longtemps à la suite dans le langage-machine d'une seule machine devient un peu asséchant, quelque chose y devient un peu raide, quelquechose qui ressemble à la tristesse d'un ordinateur qui n'executerait qu'1 ou 2 programmes, et toujours les mêmes.

Or un des buts de l'informatique est de rendre les programmeurs heureux.

Pourtant, il n'est pas contestable que la première machine sur laquelle on s'est fait les dents est plutôt déterminante sur notre point de vue ultérieur sur les langages-machines. Ya des machines qui contraignent littéralement leur utilisateur à accumuler feintes et astuces ; ça fait des esprits obliques, habiles à contourner l'oppression ; dans l'hypothèse la plus optimiste, je crois que ça fait des gens habitués à se construire, sur toute machine, LEUR machine.

Je n'aime pas vraiment cette idée. Je crois que ça rend orgueilleux, ce qui a une bonne tendance à augmenter le temps de mise au point de programmes un peu longs.

Je préfère que soit offerte au débutant une machine plus souple, orientée vers une programmation plus expressive, je veux dire, qui donne un style de programmation qui permettent d'exprimer et de décrire les problèmes que le programme traite. Bref, des programmes dont la lecture donne plus une idée du problème, que des acrobaties du programmeur.

Voilà pourquoi il y a QUASI 11.

On discute souvent de ces idées-là avec C. LENORMAND et Y. DEVILLERS. Je remercie H. WERTZ qui a detecté et rectifié une erreur dans l'interprète tout à fait insidieuse.

Si vous trouvez une erreur manifeste de fonctionnement dans le systeme, laissez tout dans l'état, et venez me prévenir. Ça permettra de faire sauter encore une des erreurs qui restent sans aucun doute dedans.

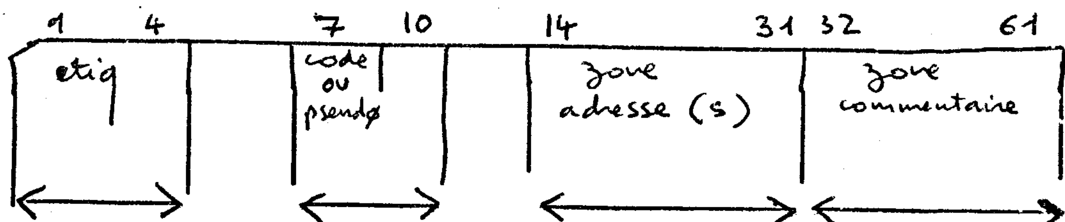
DESCRIPTIØN PLUTØT RAPIDE MAIS CØMPLETE
DE L'ASSEMBLEUR.

ETIQUETTE : c'est une étiquette définie par l'usager, d'une longueur non supérieure à 4 caractères ØU le caractère « . »

ADRESSE : c'est : etiquette
ØU etiquette ± nombre

NØMBRE : c'est un nombre.

Munis de ces indications, considérons les PSEUDØS, il y a, sachant que la mise en page s'effectue comme suit :



ØRIG { nombre / adresse : implante ce qui suit à partir de l'adresse indiquée

END { nombre / adresse : indique la fin du programme à assembler, ainsi que l'adresse de lancement.

RES nombre : reserve autant de mots que le nombre en spécifie.

DC { adresse
nombre
/ chaîne de caractères /

range l'adresse ou le nombre dans un mot, ou bien range la chaîne de caractères en mémoire, à raison de 1 caractère par mot.

C'est tout. Voici quelques remarques utiles :

- ne pas assembler en dessous de l'adresse 11.
- s'il n'y a point d'ØRIG, l'assemblage débute automatiquement à partir de l'adresse 11.
- la zone adresse de END spécifie l'adresse de la 1re instruction exécutable du programme.
- 4 caractères au plus, pour une étiquette.
- le nombre associé à RES peut être ≤ 0.
- la chaîne associée éventuellement à un DC aura une longueur d'au plus 18 caractères.
- le caractère "/" est assemblable s'il suit immédiatement le 1er "/" séparateur.
- on peut placer autant d'ØRIG qu'on veut, ou pas du tout, seul le END est obligatoire.
- un caractère "*" en 1ère colonne indique une ligne commentaire.
- les connaisseurs auront déjà remarqué que les tabulations sont identiques à celles de l'assembleur 510.

L'assembleur ne manquera pas, s'il y a lieu, de donner à l'usager les diagnostics d'erreurs que voici

*ER 1 : plusieurs fois la même étiquette en zone étiquette.
*ER 2 : code inconnu
*ER 3 : adresse inconnue
*ER 4 : programme trop long

Voici des exemples.

→

	ØRIG	60
HØP	DC	15
BØPL	RES	2
A	DC	/B/
*		

on a les liaisons etiquettes - adresses

HØP ↔ 60
BØPL ↔ 61
A ↔ 63

et on aura en mémoire :

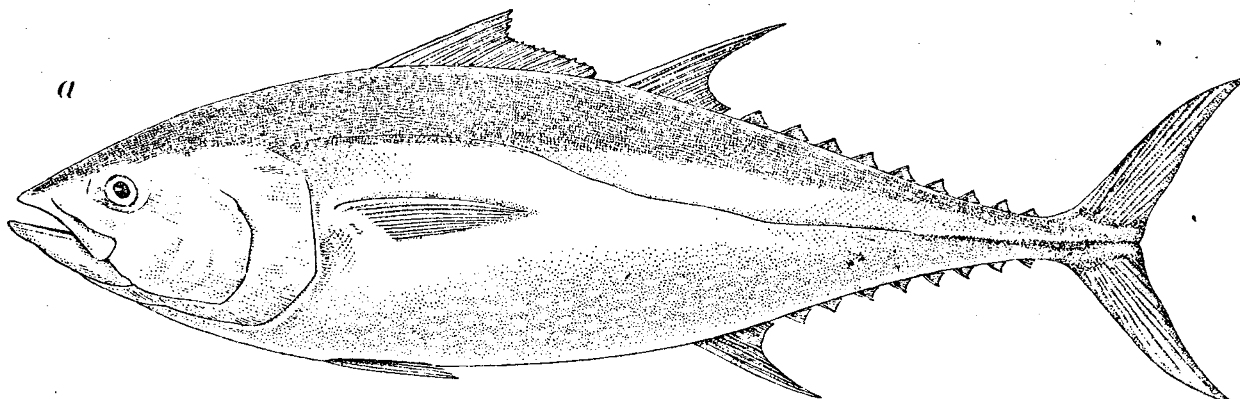
60	15
61	
62	
63	"B"

→

	ØRIG	15
NIB	DC	• + 3
FRRR	DC	NIB
	DC	/DER/
	DC	-5
*		

NIB ↔ 15
FRRR ↔ 16

15	18
16	15
17	"D"
18	"E"
19	"R"
20	-5



→
```
        ØRIG   206
NIB     RES    0
        DC     NIB+50
NØB     DC     NIB-20
*
        ØRIG   NØB+4
        DC     .-1
        DC     NØB
*
```

NIB ↔ 206
NØB ↔ 207

206	256
207	186
208	
209	
210	
211	210
212	207

```
        ØRIG   90
MESG    DC     /MESSAGE⊓ØK/
BGKP    RES    0
SEC     RES    3
QUØI    DC     .+10
*
```

MESG ↔ 90
BGKP ↔ 100
SEG ↔ 100
QUØI ↔ 103

90	M
91	E
92	S
93	S
94	A
95	G
96	E
97	⊓
98	Ø
99	K
100	
103	113

EXERCICES : indiquer a/ les associations adresses - etiquettes
b/ la repartition en memoire de

①

```
      ØRIG   20
AA    DC     /-+/
      RES    1
BB    DC     //*/
*
```

②

```
      ØRIG   600
A     RES    2
      ØRIG   A-100
B     RES    2
      ØRIG   B-100
C     DC     /s/
```

③

```
      ØRIG   110
B     DC     .+3
A     DC     A-5
      DC     B+500
      DC     .-20
```

④

```
      ØRIG   20
A     RES    100
      ØRIG   .
TABL  DC     10
      DC     -20
      DC     30
      DC     -40
```

SØLUTIØNS :

① AA ⟺ 20
BB ⟺ 23

20	21	22	23	24
-	+		/	*

② A ⟺ 600
B ⟺ 500
C ⟺ 400

400
s

③ B ⟺ 110
A ⟺ 111

110	111	112	113
113	106	611	93

④ A ⟺ 20
TABL ⟺ 120

120	121	122	123
10	-20	30	-40

DESCRIPTIØN MØINS RAPIDE, MAIS PLUS DIDACTIQUE, DE LA MACHINE DØNT CE SIMULATEUR EST LE SIMULATEUR.

Parlons un peu, à présent de l'engin. Je vais y aller en douceur ; il y a des tableaux synoptiques et éclairants à la fin du papier.

Il existe 10 registres à tout faire, numérotés de 0 à 9. Deux d'entre eux ne sont pas si à tout faire que ça, le 9 est le compteur ordinal, le 8 est un pointeur de pile. Je ne toucherai pas pour l'instant à ces deux là, et nous ferons avec 0 à 7. Je les dénote par R0, R1, ..., R7 et, une adresse en mémoire (adresse x), je l'écris M [x].

On va se limiter pour le moment à quelques instructions.

CLR D	: elle met zéro dans D.

exs :

CLR	3	effet :	0 → R3
CLR	7		0 → R7

exs :

	ØRIG	20	
BØP	RES	1	
BIP	RES	1	
⋮			
	CLR	BØP	0 → M [20]
	CLR	BIP	0 → M [21]

Heureuse machine qui permet d'adresser de la même façon mémoire et registres ! (Mais tout se paye, dans ce monde idéal, et adresser la mémoire prend, à l'exécution, plus de temps qu'adresser un registre). CLR est, à l'évidence, une instruction à 1 adresse, cette adresse je la nomme destination (D).

On en apprend une autre :

| MØV S, D | : elle place le contenu de S en D.

S pour "source"
D pour "destination", c'est tout à fait parlant.

S et D peuvent être, indépendamment, 1 registre, ou 1 mot en mémoire.

exs :

MØV	0, 1	R0 → R1
MØV	7, 5	R7 → R5
MØV	1, 1	R1 → R1 (idiot).

exs :

	ØRIG	20	
BØP	RES	1	
BIP	RES	1	
⋮			
	MØV	BØP, 2	M[20] → R2
	MØV	4, BIP	R4 → M[21]
	MØV	BIP, BØP	M[21] → M[20]

MØV est donc une instruction à 2 adresses.
Allons y d'un petit bout de programme connu

AVANT R0 [−1] R1 [40]

et : MØV 0,2
MØV 1,0
MØV 2,1

APRES R0 [40] R1 [−1]

Un autre, de la même farine :

	ØRIG	100
BØP	DC	−1
BIP	DC	40
	⋮	
	MØV	BØP, 0
	MØV	BIP, BØP
	MØV	0, BIP

AVANT :

100	101	R0
−1	40	∞

APRES

100	101	R0
40	−1	−1

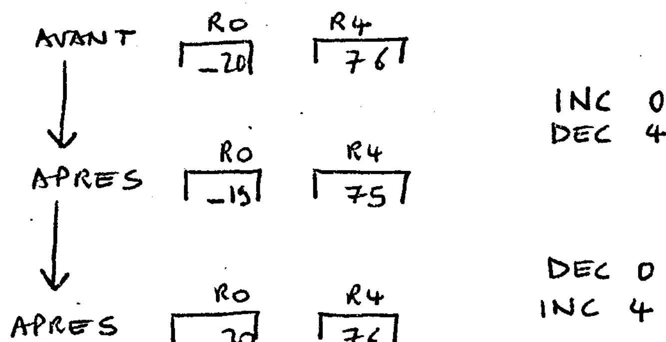
Il est clair qu'ont été échangés les contenus de BØP et BIP.

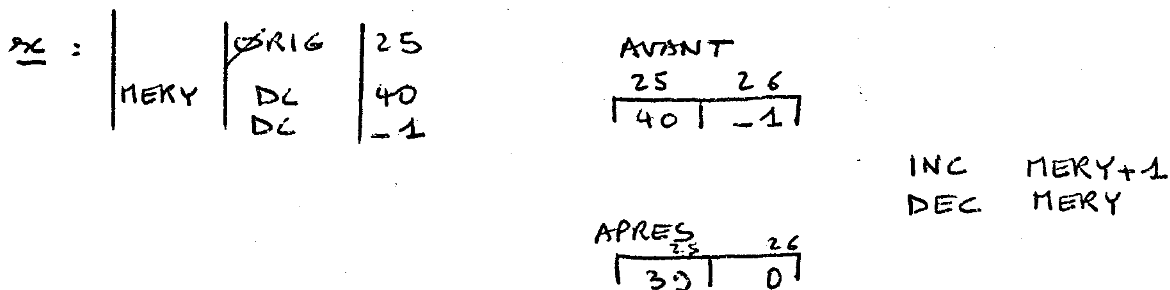
Nous connaissons 2 instructions : CLR D 0 → D
MØV S,D S → D

Connaissons-en de nouvelles :

INC D D+1 → D
DEC D D-1 → D

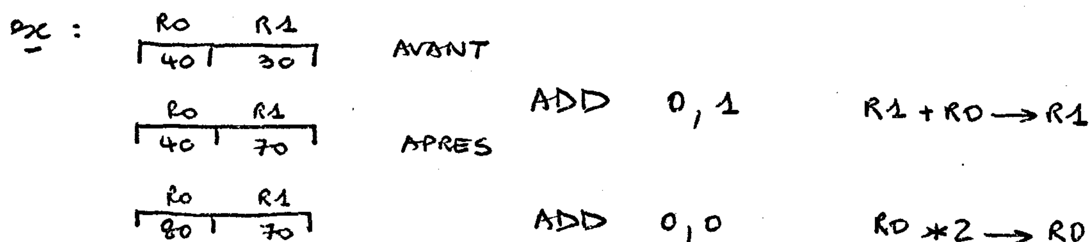
exemples :

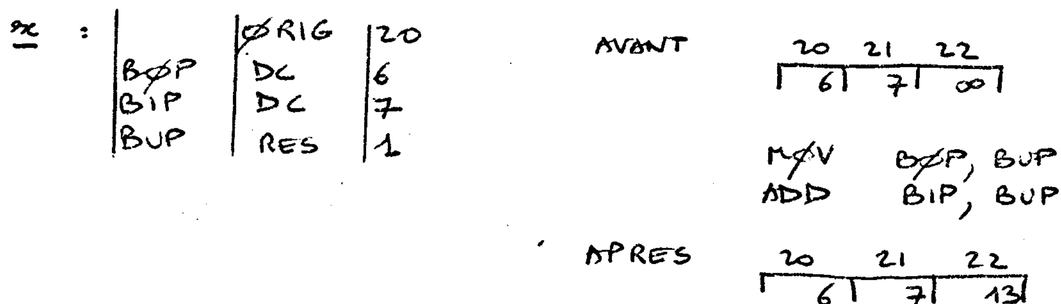




Voici encore une petite instruction à 2 adresses :

ADD S, D D + S → D





Grimpons à présent d'un cran dans l'adressage. Lorsque S ou D sont des registres, c'est un adressage par registre. Lorsque S ou D sont un mot de mémoire, c'est un adressage "relatif" (nous verrons plus tard que cela signifie relatif au compteur ordinal).

Voici à présent l'adressage immédiat : on spécifie l'opérande directement dans l'instruction, et on indique ceci par = opérande, celui-ci pouvant être un nombre tout autant qu'une adresse.

ex :

MØV	=5, 2	5 → R2
MØV	=−71, 0	−71 → R0
MØV	=15, BØP	15 → BØP

ex :

	ØRIG	100	
BØP	DC	/QUØI/	
BIP	RES	1	
	⋮		
	MØV	= BØP, 2	100 → R2
	MØV	= BØP+3, BIP	103 → M [104]

Donc, CLR 3 est une façon compacte et plus rapide d'exécution de dire : MØV =0, 3.

Avançons encore dans la connaissance de cet ordinateur récent. Nous connaissons déjà les 10 registres. Il existe également une bidouille qui peut se placer dans 6 états, non véritablement tous distincts, et bien connus des fortranistes : EQ, NE, LT, LE, GT, GE.

La dite bidouille se positionne d'elle même dans l'état de la D(estination) de l'instruction la plus récemment exécutée. Nommons la bidouille CC (Condition Code. cf le PSW du 360).

ex

	CC	R0	
AVANT	∞	1	
↓ APRES	EQ	0	DEC 0
↓ APRES	LT	−1	DEC 0

J'ai affirmé tout à l'heure que les états n'étaient pas tous distincts, en effet

EQ	veut dire	= 0
NE		≠ 0
LT		< 0
LE		≤ 0
GT		> 0
GE		≥ 0

Le cc se teste et sera utilisé par les branchements que voici : J CONDITIONNEL

BR	D	vers D	toujours (quelque soit l'état de cc)
BEQ	D		si cc = 0
BNE	D		≠ 0
BLE	D		≤ 0
BLT	D		< 0
BGE	D		≥ 0
BGT	D		> 0

ex : voici un programme qui ne fait rien de très spécial. R0 part de −30, va jusqu'à 0 et recommence indéfiniment.

```
REC | MØV | =-30,0
REM | INC | 0
    | BNE | REM
    | BR  | REC
```

On peut donc modifier l'état de cc par une instruction qui change arithmétiquement le contenu de D, ainsi que par les deux autres que voici :

TST	D	D → CC
CMP	S, D	D − S → CC	COMPARESI

exs :

→ TST 1
BEQ ici — vers ICI si R1 = 0

→ TST HEP
BLT GØ — vers GØ si M[HEP] < 0

→ CMP 1, 0
BEQ RE — vers RE si R0 = R1, i.e. si R0 − R1 = 0

→ CMP 1, 0
BGT RE — vers RE si R0 > R1, i.e. si R0 − R1 > 0

→ CMP 3, HEP
BNE RE — vers RE si M[HEP] ≠ R3.

Exercice : que fait le programme suivant ?

```
        MØV   =8,0
        MØV   =5,1
        BR    E2
E1      INC   1
        DEC   0
E2      BNE   E1
```

Solution : c'est une façon tout à fait récurrente d'effectuer l'addition $8+5=13 \rightarrow R1$

Exercice : que fait ceci ? :

```
        MØV   =11,3
        CLR   1
        BR    TEST
RE      ADD   3,1
TEST    DEC   3
        BGT   RE
```

Solution : ceci place dans R1

$10+9+8+7+\ldots+2+1=55$

Exercice : et ceci ?

```
        CLR   0
        MØV   =1,1
        MØV   =7,2
        BR    RE2
RE1     MØV   1,3
        ADD   0,1
        MØV   3,0
RE2     DEC   2
        BNE   RE1
```

Solution : est placé dans R1 : 13, 7ème élément de la suite de Fibonacci.

Sans nul doute, le lecteur se sent à présent détendu et parfaitement à son aise. Dès lors, continuons à approfondir ces questions d'adressages. Nous en connaissons jusqu'à présent trois :

1°/ adressage par registre, S ou D $\in [0, 9]$
2°/ relatif, S ou D adresse en mémoire
3°/ immédiat, S ou D : = opérande :

En voici un nouveau, l'adressage indexé.

S ou D : adresse (registre)

exemple :

```
        ØRIG   100
TAB     DC     1
        DC     2
        DC     3
        RES    1
*
        MØV    =2,0
*
        MØV    TAB(0),1
        INC    0
        MØV    =4,TAB(0)
        ADD    TAB(0),TAB
```

100 101 102 103 : 1 | 2 | 3 | ∞

R0 : 2

M [100 + 2] → R1 — R1 : 3

R0 : 3

4 → M [100 + 3] — 100 101 102 103 : 1 | 2 | 3 | 4

100 101 102 103 : 4 (5) | 2 | 3 | (4)

/ TU AURAIS MOV TAB(0),TAB

L'opérande sera donc, si S ou D : ADR (reg)

M [ADR + valeur du registre]

exemple : remettre à 0 une table de 10 éléments.

```
        ØRIG   20
*
TTAB    RES    10
*
        MØV    =9,1
RE      CLR    TTAB(1)
        DEC    1
        BGE    RE
```

20 21 22 23 24 25 26 27 28 29

R1 : 9

R1 - 1 [20 + 1] CLR TAB(1)

exemple : une autre façon de faire la même chose :

```
        ØRIG   20
TTAB    RES    10
*
        MØV    =-10,1
RE      CLR    TTAB+10(1)
        INC    1
        BNE    RE
```

20 21 22 23 24 25 26 27 28 29

0 ↑ ↑ i+1

R1 : -10

[20+10] = [30 + 10] = [22]

30 + 9 = 21

exemple : copier un vecteur de 80 elements dans un autre vecteur de 80 elements.

```
        ØRIG    70
*
VEC1    RES     80
VEC2    RES     80
*
        MØV     =79,0
RE      MØV     VEC1(0),VEC2(0)
        DEC     0
        BGE     RE
```

exemple : placer 80 caractères "␣" (blanc) dans une zone de 80 elements.

```
        ØRIG    30
*
ZØNE    RES     80
BLAN    DC      /␣/
*
        MØV     =-80,0
RE      MØV     BLAN,ZØNE+80(0)
        INC     0
        BNE     RE
```

exemple : la même chose, legerement plus rapide.

```
        MØV     =-80,0
        MØV     BLAN,1
RE      MØV     1,ZØNE+80(0)
        INC     0
        BNE     RE
```

exemple : additionner 2 vecteurs V1, V2 dans un troisième.

```
        ØRIG    200
*
V1      RES     30
V2      RES     30
V3      RES     30
*
        MØV     =29,0
RE      MØV     V1(0),V3(0)
        ADD     V2(0),V3(0)
        DEC     0
        BGE     RE
```

exemple : produit scalaire des vecteurs V1 et V2 de 30 elements (sachant que MUL S, D effectue D*S → D)

i.e. $\sum_{i=0}^{24} V1[i] * V2[i] \rightarrow PSCA$.

```
        ØRIG   15
*
V1      RES    15
V2      RES    15
*
PSCA    RES    1
*
        MØV    =14,0
        CLR    1
RE      MØV    V1 (0),2
        MUL    V2 (0),2
        ADD    2,1
        DEC    0
        BGE    RE
        MØV    1, PSCA
```

exemple : ordonner par ordre croissant une suite SV de 20 entiers (Tri de BUBBLE)

```
        ØRIG   40
SV      RES    20
*
        MØV    =19,0

E1      MØV    0,1
E2      DEC    1
        CMP    SV (0), SV (1)
        BLE    E3
        MØV    SV (0), 2            sv(0) → R2
        MØV    SV (1), SV (0)       sv(1) → sv(0)
        MØV    2, SV (0) [1] errata R2 → sv(0)
E3      TST    1
        BNE    E2
        DEC    0
        BNE    E1
```

exemple : étant donné une suite de 50 elements, placer dans MAX l'element maximum de la suite.

```
        ØRIG   30
*
SUIT    RES    50
MAX     RES    1
*
        MØV    SUIT+49, MAX
        MØV    =48, 0
E1      CMP    MAX, SUIT (0)
        BLE    E2
        MØV    SUIT (0), MAX
E2      DEC    0
        BGE    E1
```

Maintenant que nous sommes familiers avec tout ce qui précède, prenons de la hauteur. Voici une table de toutes les instructions, à l'exception toutefois

- des entrées/sorties que nous allons voir tout à l'heure
- des instructions de branchement et retour à (de) sous-programmes.

INSTRUCTIØNS :

à zérø adresse :	HALT			termine l'exécution du programme, et fait imprimer la durée de cette exécution.
à une adresse :	BR	D	:	vers D
	BNE	D	:	vers D si CC ≠ 0
	BEQ	D	:	= 0
	BGT	D	:	> 0
	BGE	D	:	⩾ 0
	BLT	D	:	< 0
	BLE	D	:	⩽ 0
	CLR	D	:	0 → D
	INC	D	:	D+1 → D
	DEC	D	:	D−1 → D
	NEG	D	:	−D → D
	TST	D	:	D → CC
à deux adresses :	MØV	S, D	:	S → D
	CMP	S, D	:	D − S → CC
	ADD	S, D	:	D + S → D
	SUB	S, D	:	D − S → D
	MUL	S, D	:	D ∗ S → D
	DIV	S, D	:	D ÷ S → D

Jeu d'instructions calme et sans problème, tout à fait indiqué pour l'enseignement de l'algorithmique non-numérique et combinatoire.

Avant de découvrir les traits les plus avancés de cette honorable machine, soyons pratique en considérant ici les problèmes d'entrées et de sorties de données.

les choses entrent SUR CARTES.
sortent (sont imprimées) SUR DU PAPIER.

Nous pouvons entrer (et sortir)
. des chaînes de caractères alphanumériques.
. des suites d'entiers (cadrés et convertis).

Voyons un peu :

ENTRER DU CARACTERE : RA D

sont aussitôt (voire !) avalés les 80 caractères alphanumériques à partir de l'adresse D en mémoire.

exemple :

```
       ØRIG  100
ZØNE   RES   80
        :
       RA    ZØNE
```

100	L
101	E
102	⊓
103	C
104	I
105	E
106	L
107	⊓
:	:

```
 1 2 3 4 5 6 7 8
 L E ⊓ C I E L ⊓ ...
```

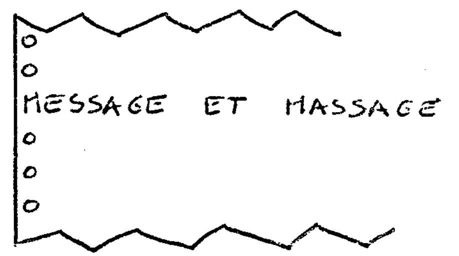
SØRTIR DES CHAÎNES : WA S, D

cette instruction fait imprimer les S caractères en mémoire à partir de l'adresse D.

exemple :

```
       ØRIG  200
MESG   DC    /MESSAGE⊓ET⊓MASSAGE/
        :
       WA    =18, MESG
```

ceci imprime (on s'en doutait) :



exemple : sauter une ligne,

```
BLAN   DC    /⊓/
        :
       WA    =1, BLAN
```

LIRE UNE SUITE D'ENTIERS : RI S,D

lit sur une carte S entiers, $S \leq 10$, et les place en mémoire à partir de l'adresse D. Sur la carte, les entiers doivent être cadrés à droite toutes les 8 colonnes (i.e. le format 10 I 8 Fortran, pour ceux qui aiment ça).

exemple :

```
        ØRIG   20
*
TAB     RES    4
*
        MØV    =4,0
        RI     0,TAB
```

et ceci avale, par exemple :

8	16	24	32
40	-5	3	267

et place en mémoire :

20 | 40
21 | -5
22 | 3
23 | 267

(J'aurai, bien entendu, pu écrire aussi : RI =4,TAB).

ECRIRE UNE SUITE D'ENTIERS : WI S,D

écrit sur une même ligne S entiers, $S \leq 10$, stockés en mémoire à partir de l'adresse D. Ils sont également cadrés en I8 sur la ligne.

exemple :

```
        ØRIG   30
*
        DC     -8
        DC     5
        DC     7
BTAB    DC     6
*
        :
        WI     =4,BTAB-3
```

Ceci écrit :

-8 5 7 6

Comme vous le savez, ça prend un temps fou, les entrées/sorties. Donc, si par exemple, on entre des données qu'on veut immédiatement traiter, il y a intérêt à ce que le processus de lecture soit terminé quand on commence le traitement, sans quoi la donnée ne sera généralement pas encore arrivée là où on la traite, et on n'aurait pas l'air malin.

Bref, il faut se donner les moyens d'attendre que l'opération d'entrée ou de sortie soit terminée. Attendre sur place, que ça se passe, ou bien en profiter pour faire avancer le travail dans une autre zone du programme, qui ne dépend pas des données à lire ou à écrire. Ce moyen vous est donné par les instructions que voici.

BBR D : vers D si l'opération de lecture en cours n'est pas achevée.

BBW D : vers D si l'opération d'écriture en cours n'est pas achevée.

exemple : je veux lire un entier dans NB, lui ajouter 1, puis l'imprimer, puis arrêter.

1ER ESSAI

```
        ØRIG   30
NB      RES    1
*
DEB     RI     =1,NB        1
        INC    NB           2
        WI     =1,NB        3
        HALT                4
        END    DEB
```

ligne 1 : ceci déclenche une opération de lecture de 1 nombre à partir de NB, et ça passe à la suite.

ligne 2 : ceci incrémente le contenu de NB, mais évidemment, le nombre n'est pas encore arrivé dans NB. RATÉ !

2D ESSAI :

```
DEB   RI    =1,NB    1
      BBR   .        2
      INC   NB       3
      WI    =1,NB    4
      HALT           5
```

ligne 1 : déclenche une opération de lecture de 1 nombre à partir de NB et passe à la suite.

ligne 2 : on boucle sur la ligne 2 tant que dure l'opération de lecture, et quand c'est fini, on va en

ligne 3 : on incrémente le nombre.

ligne 4 : on déclenche une opération de ~~lecture~~ ECRITURE de 1 nombre à partir de NB, et on passe à la suite.

ligne 5 : où l'exécution du programme s'achève, et rien n'a été imprimé. ENCØRE RATÉ !!

3ÈME ESSAI (et le bon) :

```
DEB   RI    =1,NB
      BBR   .
      INC   NB
      WI    =1,NB
      BBW   .
      HALT
```

Est-ce vu ?

Bon ; il y a encore quelques aspects idiomatiques qu'il n'est pas mauvais de savoir.

Si on passe sur un BBR ou un BBW, alors qu'il n'y a pas d'opération de lecture (resp. d'écriture) en cours, ça n'a strictement aucun effet. Il ne faut donc pas se gêner.

Si, une opération d'I/Ø étant en cours, on passe sur un autre RI (RA) ou WI (WA), il est alors (ce dernier) complètement ignoré.

exemple : rentrer 100 nombres dans la table TAB

```
        ØRIG   20
*
TAB     RES    100
*
        MØV    =10,0
        MØV    =-100,1
RE      RI     0,TAB+100(1)
        BBR    .
        ADD    0,1
        BNE    RE
```

Il est clair qu'avec ces primitives un peu rustiques (BBR et BBW) on peut faire et enseigner toutes les excellentes bricoles, buffering, buffer swapping, buffers multiples, coroutines etc. décrites, par exemple dans les pages 211-225 du tome 1 du monument de Donald KNUTH *.
Le débutant pourra donc, sans trop de frais, éprouver les fortes sensations consécutives à l'écrasement de ses buffers.

* Donald E. KNUTH (1968) : The Art of Computer Programming. Vol 1 : Fundamental Algorithms. ADDISØN-WESLEY.

Après cet intermède comique, tournons-nous résolument vers l'avenir, et considérons quelques aspects moins folkloriques de notre engin.

Au fond, je crois que ce qui est important dans les machines de 3ème génération c'est

1°/ une bordée de registres à tout faire, sans distinction bidon entre registres d'index et totalisateurs.

2°/ un jeu d'adressage complet et intelligent.

Nous connaissons jusqu'à présent quatre modes d'adressage de notre bestiole.

1°/ par registre	exemple :	INC 0
2°/ relatif		MØV ADR, 3
3°/ IMMEDIAT		MØV = -1, 7
4°/ INDEXÉ		INC TTAB+71 (4)

Pour chacun de ces 4 modes, il existe deux aspects dont nous n'avons vu jusqu'à présent que le premier.

1. DIRECT

2. INDIRECT

exemple : BR RE : je m'envoie en RE

mais : [MØV =RE, 1 / BR @1] : je m'envoie en l'adresse qui est la valeur courante de R1, ici RE.

le signe "@" indique que l'adressage, quelque soit son mode est alors INDIRECT.

exemple : je veux écrire quelque chose comme ça :

si R0 = 0 vers E1 sinon
si R0 = 1 vers E2 sinon
si R0 = 2 vers E3 sinon vers E4 (sachant que R0 ∈ [0, 3])

```
       ØRIG   30
ETIQ   DC     E1
       DC     E2
       DC     E3
       DC     E4
*
       MØV    ETIQ (0), 3
       BR     @3
```

exemple :

	ØRIG	30
BØP	DC	HEP
HEP	RES	1
*	MØV	=1, @BØP

: place 1 dans HEP
i.e. 1 → M[31]
ou encore : 1 → M[M[30]]

Tres pratique, l'indirection, dans tous les cas où il y a à manipuler des listes, des aiguillages, des chaînes, et en général des adresses et des paramètres.

Bien sûr, les personnes à l'esprit compliqué auront déjà repéré d'apparentes incohérences inhérentes à ces modes d'adressage.

exemples de stupidités particulières :

MØV =1, =0 , ce qui à l'air de ne rien vouloir dire, pas plus que ce qui suit :

MØV @=3, 1 ou encore ceci

MØV 2, =ADR

exemples de redondances :

MØV ADR, 0 est équivalent à

MØV @=ADR, 0 cette dernière étant toutefois d'execution plus longue.

Ø SAISØNS , Ø CHATEAUX .

Nous verrons que tout cela se résout finalement très bien, et les mêmes gens compliqués y verront là une occasion supplémentaire d'astucer tout à leur aise ; c'est après tout leur problème s'ils aiment rendre, de leurs programmes, les mises au point interminables.

Voici deux nouvelles possibilités d'adressage, de puissance considérable.

1°/ mode AUTØ-INCREMENT

S ou D (reg)+

. C'est tout à fait identique à @reg, i.e. l'adresse d'opérande est le contenu du registre reg, mais après extraction de l'opérande, le registre reg est incrémenté de 1.

exemple : MØV (1)+, 0

est équivalent à
[MØV @1, 0
[INC 1

2°/ mode AUTØ-DECREMENT

S ou D -(reg)

. C'est la même chose que 1°/, mais, avant l'extraction, le registre est décrémenté de 1.

Donc, MØV -(1), 0 est équivalent à :

[DEC 1
[MØV @1, 0

Ceci donne le moyen de manipuler des PILES sans trop se fatiguer, le registre reg servant alors de pointeur de sommet de pile. Les graphies (reg)+ et -(reg) soulignent assez les moments où le registre pointeur est incrémenté (resp. decrémenté) i.e. avant ou après l'extraction de l'opérande.

Bien sûr, cette possibilité remarquable servira à bien d'autres usages également que les manipulations de piles, nous verrons cela bientôt.

Voici à présent un exemple, certes un peu forcé d'utilisation du dispositif.

```
          ØRIG    20
RESU      RES     1
          ØRIG    199
PILE      RES     1
*
          MØV     =PILE+1, 1          R1 pointeur de pile.
          MØV     =RESU, -(1)         empile l'adresse RESU
          MØV     =5, -(1)            empile l'entier 5
          MØV     =2, -(1)            empile l'entier 2
          MØV     =3, -(1)            empile l'entier 3
```

état de la pile

197	3 ← R1
198	2
199	5
200	20

```
          MUL     (1)+, @1
```

ceci effectue : { M [R1] → opd1 ; R1 + 1 → R1 ; M[R1] * opd1 → M [R1] }

etat de la pile

198	6 ← R1
199	5
200	20

```
          ADD     (1)+, @1
```

même chose.
état de la pile :

199	11 ← R1
200	20

```
          MØV     (1)+, @(1)+
```

ceci effectue :
{ M [R1] → opd1 ; R1 + 1 → R1 ; M [R1] → opd2 ; R1 + 1 → R1 ; opd1 → M [opd2] }

outrement dit 11 → M [RESU]
et R1 = 201

pile 200 ← R1

le lecteur aura noté qu'il s'agit de l'execution de l'expression post-fixée : RESU 5 2 3 * + ←

Les gens qui étudient et produisent des compilateurs voient d'ici les avantages que leur offre un adressage pareil. On y compilera a priori du code épatant.
Il y a aussi une foultitude d'autres possibilités, je vais en énumérer quelques unes ; reste qu'il ne faut pas bien sûr balancer ça tel quel aux débutants ; avec les modes registre, immédiat, relatif et indexé, ils ont déjà de quoi s'occuper pour un moment.

Il y a eu en 1970, dans les actes de la Spring Joint Computer Conference *, un papier sur la machine inspiratrice de notre engin, et, dedans, il y a quelques joyeux exemples que je vous livre tels quels.

FØRMAT 2 ADRESSES :

MØV	B, A	B → A	placer contenu de B en A
MØV	=N, A	N → A	placer N dans A
MØV	B(1), A(1)	B[i] → A[i]	vecteur → vecteur
MØV	(1)+, (2)+	{ B[i] → A[i] ; i+1 → i	vecteur → vecteur et avancer

FØRMAT REGISTRES :

MØV	A, 1	A → R1	load
MØV	1, A	R1 → A	store
MØV	@A, 1	M[A] → R1	load indirect
MØV	1, 3	R1 → R3	inter registres
MØV	1, A(2)	R1 → A[i]	store indexé

* G. BELL et al. (1970) : "A new architecture for mini-computers, the DEC PDP-11"
Proc SJCC 657-675

MØV	@A(0), 1	M[A[i]] → R1	load indexé indirect
MØV	@0, 1	M[R0] → R1	load indirect via registre.
MØV	(1)+, 3	{ M[R1] → R3 R1 + 1 → R1	load indirect via registre et avancer.

FØRMAT MACHINE A PILE : S : sommet de pile.

MØV	=N, -(0)	empiler N	N → S
MØV	A, -(0)	empiler contenu de A	A → S
MØV	@(0)+, -(0)	M[S] → S	empiler le contenu de l'adresse stockée en sommet de pile.
MØV	(0)+, A	S → A	depiler dans A
MØV	(0)+, @(0)+	S → M[S]	placer le sommet de pile à l'adresse empilée en sommet de pile - 1
MØV	@0, -(0)	S → S	duplique le sommet de pile.
[TST BNE	@0 ETIQ	vers ETIQ si sommet de pile ≠ 0	
NEG	@0	-S → S	

C'est confortable, sans feintes ni astuces. Seulement utilisation ordinaire d'un adressage intelligent.

exemples :

. initialiser à 0 1 vecteur de 100 elements (plus rapide que tout à l'heure.

```
        ØRIG   20
VECT    RES    100
*
        MØV    =-100,0
        MØV    =VEC,1
RE      CLR    (1)+
        INC    0
        BNE    RE
```

. un vecteur de 30 elements. Produit de convolution dans RESU, i.e. $\sum_{i=0}^{29} V[i] * V[29-i] \longrightarrow$ RESU

```
        ØRIG  64
V       RES   30
RESU    RES   1
*
        MØV   =-30,0
        MØV   =V+30,1
        CLR   2
*
RE      MØV   -(1),3
        MUL   V+30(0),3
        ADD   3,2
        INC   0
        BNE   RE
```

. coller dans LØNG l'adresse du dernier caractère non-blanc d'un buffer BUF

```
        ØRIG  77
BUF     RES   80
LØNG    RES   1
BLAN    DC    /␣/
*
MØ      MØV   BLAN,0
        MØV   BUF+80,1
*
RE      CMP   0,-(1)
        BNE   RE
        INC   1
        MØV   1,LØNG
```

. tester si le contenu d'1 vecteur MIR de 15 elements est ou non un mot miroir. i.e. si $\left|\begin{array}{l} i \in [0,7] \\ MIR[i] = MIR[15-i] \end{array}\right.$

```
        ØRIG  305
MIR     RES   15
*
        MØV   =MIR+15,0
        MØV   =MIR,1
RE      CMP   (1)+,-(0)
        BNE   NØN
        CMP   1,0
        BNE   RE
ØUI     ...

NØN     ...
```

```
*
*       MOT DE DYCK
*
ZONE    RES     80
F       DC      /F/
BLAN    DC      / /
LPAR    DC      /(/
RPAR    DC      /)/
YES     DC      /BON/
NO      DC      /MAUVAIS/
*
DEBU    RA      ZONE
        BBR     .
        WA      =80,ZONE
        BBW     .
        CMP     ZONE,F
        BNE     SUIT
        HALT
*
SUIT    CLR     1                   R1 COMPTEUR
        CLR     0                   R0 INDEX
LOOP    CMP     ZONE(0),BLAN
        BEQ     VOIR
        CMP     ZONE(0),LPAR
        BEQ     INCR
        CMP     ZONE(0),RPAR
        BNE     AVAN
*
        DEC     1
        BLT     NON
        BR      AVAN
*
INCR    INC     1
AVAN    INC     0
        BR      LOOP
*
VOIR    TST     1
        BGT     NON
*
        WA      =3,YES
        BR      WAIT
NON     WA      =7,NO
WAIT    BBW     .
        BR      DEBU
*
        END     DEBU
```

```
(()())()
BON
((()
MAUVAIS
(()))
MAUVAIS
((((()))())
BON
F
```

1292 UNITS

```
*
*       PERMUTATION INVERSE DE 0 ... N-1 AVEC
*       MEMOIRE AUXILIAIRE.
*       N INF OU EGAL A 10
*
*
        ORIG    100
LPER    RI      N,XPER
        BBR     .
        WI      N,XPER
*
        MOV     N,5             R5 INDEX XPER
RE      DEC     5
        MOV     XPER(5),4
        MOV     5,YPER(4)       R4 INDEX YPER
        BNE     RE
*
        BBW     .
        WI      N,YPER
        BBW     .
*
READ    RI      =1,N
        BBR     .
        WI      =1,N
        TST     N
        BNE     LPER
        BBW     .
        HALT
*
N       RES     1
XPER    RES     10
YPER    RES     10
*
        END     READ
```

5								
2	3	4	0	1				
3	4	0	1	2				
3								
0	2	1						
0	2	1						
9								
4	8	0	7	1	5	3	6	2
2	4	8	6	0	5	7	3	1
0								

778 UNITS

```
*
*       PERMUTATION INVERSE SANS MEMOIRE AUXILIAIRE.
*
        ORIG    20
*
LPER    RI      N,PERM+1
        BBR     .
        WI      N,PERM+1
*
        MOV     N,1                     R1 INDEX
        BBW     .
E1      NEG     PERM(1)
        DEC     1
        BGT     E1
*
        MOV     N,1
E2      MOV     1,3
        NEG     3
E3      MOV     3,2
        NEG     2
        MOV     PERM(2),3
        NEG     3
        BLT     E3
*
        MOV     PERM(3),PERM(2)
        MOV     1,PERM(3)
        DEC     1
        BGT     E2
*
        WI      N,PERM+1
*
READ    RI      =1,N
        BBR     .
        WI      =1,N
        TST     N
        BNE     LPER
        BBW     .
        HALT
*
N       RES     0
PERM    RES     11
*
        END     READ
```

4								
3	1	2	4					
2	3	1	4					
6								
4	3	5	6	2	1			
6	5	2	1	3	4			
9								
3	7	8	9	2	4	5	1	6
8	5	1	6	7	9	2	3	4
0								

1142 UNITS

```
*
*
*       RACINE CARREE APPROCHEE PAR SOUSTRACTIONS SUCCESSIVES.
*
BLAN    DC      / /
*
RAC2    CLR     1
        CLR     2
        MOV     3,0
        BR      E2
E1      INC     1
        INC     2
E2      INC     2
        SUB     2,0
        BGE     E1
*
        BBW     .
        WI      =1,1
        BBW     .
        WA      =1,BLAN
*
READ    RI      =1,3
        BBR     .
        WI      =1,3
        TST     3
        BNE     RAC2
        BBW     .
        HALT
*
        END     READ
```

9
3

20
4

8100
90

150
12

0

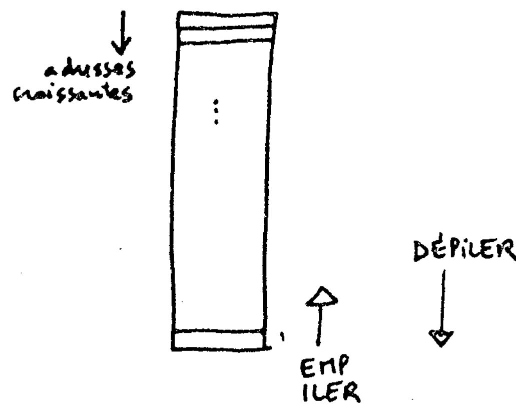
1281 UNITS

?

SØUS-PRØGRAMMES ET ENCHAÎNEMENTS DE SØUS-PRØGRAMMES

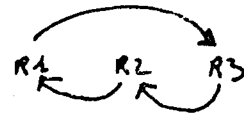
Il y a quelques temps, j'ai raconté que le registre R8 était réservé. A quoi ? Voici l'idée : quand vous faites une pile, vous vous donnez un registre pointeur de pile, et vous l'initialisez à une adresse quelconque en mémoire, qui vous sert de fond de pile.

R8, c'est la même chose, sauf que ya pas à l'initialiser. Imaginez-vous la mémoire de votre machine comme une pile, où on empile dans le sens des adresses décroissantes. L'adresse du fond de pile étant la + haute adresse en mémoire, plus 1.



exemple : une petite transposition R1 ⟷ R2.
On choisit normalement un registre auxiliaire, disons R3, et on fait :

```
MØV   1,3
MØV   2,1
MØV   3,2
```



On pouvait s'épargner l'usage de ce registre auxiliaire comme ça :

```
MØV   1,-(8)     empiler R1
MØV   2,1        R2 → R1
MØV   (8)+,2     dépiler dans R2
```

Ya pas à s'occuper de l'initialisation de R8, c'est le système qui fait ça tout seul.

Autre exemple : vous voulez ranger R1, R2, R3 dans un coin tranquille et les récupérer plus tard.

Première solution : vous réservez des places en mémoire

```
*
AUX1    RES   1
AUX2    RES   1
AUX3    RES   1
*
        :
        MØV   1,AUX1  ]
        MØV   2,AUX2  ] ranger
        MØV   3,AUX3  ]
        :
        MØV   AUX1,1  ]
        MØV   AUX2,2  ] récupérer
        MØV   AUX3,3  ]
```

Seconde solution : sans réservation de mémoire auxiliaire

```
        MØV   1,-(8)    empiler R1         ]
        MØV   2,-(8)    empiler R2         ] ranger
        MØV   3,-(8)    empiler R3         ]
        :
        MØV   (8)+,3    dépiler dans R3    ]
        MØV   (8)+,2    dépiler dans R2    ] récupérer
        MØV   (8)+,1    dépiler dans R1    ]
```

Ça va plus vite, et ça fait autant de sous en moins.

Encore un exemple : vous voulez faire ça

R0 / R1 → quotient (entier) dans R0 et reste dans R1

soit R0 = x, R1 = y

```
        MØV   0,-(8)    empiler x
        DIV   1,0       x/y → R0
        MUL   0,1       (x/y)*y → R1
        SUB   (8)+,1    R1 - x → R1
        NEG   1         - R1 → R1
```

On est en droit de considérer ça comme pratique.

Venons-en à présent à notre sujet, les SØUS-PRØGRAMMES.

Y a deux problèmes bien connus :

1/ où placer l'adresse de retour ?

2/ comment passer des arguments au sous-programme appelé ?

Je vais répondre à ces questions dans un instant.

D'abord l'instruction de branchement vers sous-programme, c'est :

JSR reg, D

i.e. vers sous-programme, dont la première instruction est à l'adresse D. Voici ce que ça fait exactement

exemple : JSR 1, SPRG

1/ ça empile R1

2/ ça colle dans R1 l'adresse de retour i.e. l'adresse de l'instruction qui suit l'appel.

3/ ça colle "SPRG" dans le compteur ordinal (je vous rappelle que c'est le registre R9).

En fait c'est équivalent à (exécuté en une seule instruction)

```
MØV   1, -(8)
MØV   9, 1
MØV   =SPRG, 9
```

Pour sortir d'un sous-programme, on fera ceci :

RTS reg

1/ ça colle le contenu du registre reg dans le compteur ordinal

2/ ça dépile dans reg l'ancien contenu de ce registre (empilé par le JSR).

exemple : RTS 1 , ça fait (en 1 seule instruction)

```
MØV   1, aux
MØV   (8)+, 1
MØV   aux, 9
```

Donc vous voyez que l'adresse de retour n'est pas, à proprement parler, empilée, elle est collée dans reg, qu'on appelle "registre de liaison", et c'est l'ancienne valeur de reg qui est empilée.

En fait, pourquoi ne pas empiler directement l'adresse de retour ? En pratique, vous pouvez le faire en utilisant le compteur ordinal comme registre de liaison

exemple : JSR 9, SP ①

et revenir par : RTS 9 ②

ça fait : par ① empiler R9 (qui contient l'adresse de retour)
puis : R9 ← adresse de retour
puis : vers SP

par ② : R9 → R9 (ce qui ne fait rien)
puis : dépiler dans R9

En pratique, on utilise R9 comme registre de liaison quand y a pas d'arguments à passer au SP (Sous-Programme). Dans ce cas, aucun autre registre que le compteur ordinal (R9) n'est modifié par l'appel.

Mais on DØIT utiliser un autre registre que R9 comme registre de liaison s'il y a à passer des arguments au sous-programme. exemple : un SP qui revient avec, en R0, la somme des contenus de 2 adresses X1 et X2 en mémoire

```
appel :             JSR   5, SØM2
                    DC    X1
                    DC    X2

sous-programme : SØM2  MØV   @(5)+,0
                       ADD   @(5)+,0
                       RTS   5
```

Voilà l'idée. Supposons que X1, c'est l'adresse 40
et que X2, c'est l'adresse 153

et [40: α] [153: β]

Par commodité d'explication, je réécris appel et SP ainsi étiquetés

```
appel :   E1     JSR   5, SØM2
          E2     DC    X1
          E3     DC    X2
          E4     ...

SP        SØM2   MØV   @(5)+,0
          E5     ADD   @(5)+,0
          E6     RTS   5
```

Donc en E1 : appel. ça place "E2" dans R5 et ça va en SØM2
en SØM2 : ça place M[M[E2]] = M[X1] = M[40] = α dans R0, et ça colle E3 dans R5 (auto-incrément).
en E5 : ça ajoute à R0 M[M[E3]] = M[X2] = M[153] = β, i.e. α+β et ça colle E4 (l'adresse de retour) dans R5.
en E6 : ça envoie à l'adresse contenue dans R5, i.e. E4 qui est précisément l'adresse de retour.

Voilà.

exemple : un sous-programme qui remet à zéro un vecteur

```
appel :          JSR.  6, RAZV
                 DC    V1          : adresse du vecteur
                 DC    10          : sa longueur
                 ...

SP      RAZV     MØV   0,-(8)      ] rangement de R0 et R1
                 MØV   1,-(8)      ]
                 MØV   (6)+,0      adresse du vecteur dans R0
                 MØV   (6)+,1      longueur dans R1
        RAZ1     CLR   (0)+        ]
                 DEC   1           ] boucle de remise à zéro
                 BNE   RAZ1        ]
                 MØV   (8)+,1      ] restitue R1 et R0
                 MØV   (8)+,0      ]
                 RTS   6           retour
```

exemple : un sous-programme NCAR qui place le nombre en R0, converti en une chaîne de caractères, à partir de l'adresse ADBF.

```
appel :          JSR   4, NCAR
                 DC    ADBF
                 ...

SP :    NCAR     MØV   (4)+,5
        NC2      MØV   =10,1
                 JSR   4,IDIV
                 MØV   1,-(8)
                 TST   0
                 BEQ   ØK
                 JSR   4,NC2
        ØK       MØV   (8)+,0
                 MØV   CHIF(0),(5)+
                 RTS   4
        CHIF     DC    /0123456789/
        IDIV     MØV   0,-(8)
                 DIV   1,0
                 MUL   0,1
                 SUB   (8)+,1
                 NEG   1
                 RTS   4
```

Les restes des divisions successives par 10 sont empilés, puis placés en ordre inverse à partir de l'adresse passée en argument à l'appel de NCAR.

Cette routine rappelle une partie d'elle-même (à l'adresse NC2) jusqu'à ce qu'un quotient nul soit obtenu en R0, par la routine de division entière IDIV (R0/R1, quotient dans R0, reste dans R1). A chaque itération, R0 est divisé par 10, le quotient résultant est placé dans R0, et le RESTE est empilé. La pile contient donc : les restes, les adresses de retour des appels de NC2, au moment où le quotient devient nul ; on va alors se brancher vers ØK.

le segment de sous-programme débutant en ØK dépile les restes, et les envoie à partir de l'adresse passée en argument à l'appel de NCAR, jusqu'à ce que la pile revienne en l'état où elle était après l'appel de NCAR. A ce moment, RTS 4 provoque le retour au programme principal.

Étudiez très soigneusement cette routine coton.

Autre exemple : un sous programme FACT qui calcule et ramène ds R0, la factorielle du nombre passé en argument dans R0. On utilisera ici R9 comme registre de liaison.

```
FACT    TST   0
        BNE   FACZ
        MØV   =1,0
        RTS   9
FACZ    MØV   0,-(8)
        DEC   0
        JSR   9, FACT
        MUL   (8)+,0
        RTS   9
```

On notera que ce SP est totalement équivalent, dans sa formulation à la fonction récursive :

```
(DE   FACT  (N)
   (CØND
      ((ZERØP N) 1)
      (T (TIMES  N  (FACT  (SUB1  N )) )) ))
```

Je m'en voudrais de ne pas terminer ce papier sans souligner une possibilité remarquable, dont le constructeur DIGITAL est particulièrement fier (quoique j'ai des raisons de penser que cette possibilité, conséquence de l'adressage, n'a pas été à proprement parler pensée à l'avance).

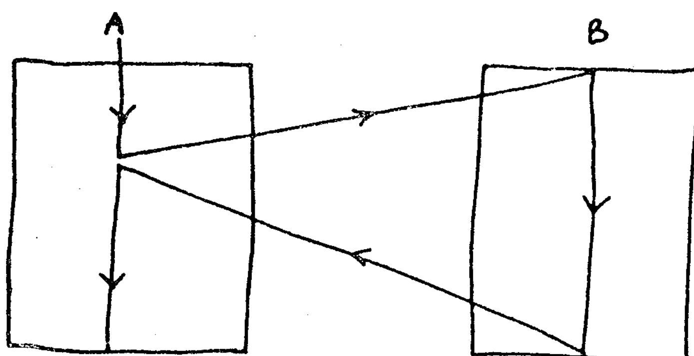
Si je fais :

JSR 9, @(8)+

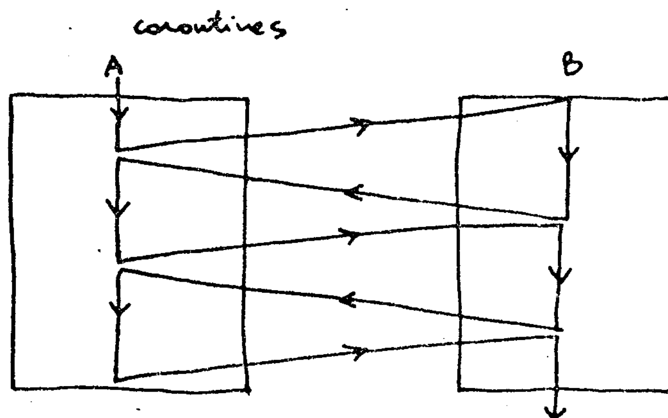
ça échange le SØMMET DE PILE et le compteur ordinal.

ça sert à 2 sous-programmes pour s'appeller mutuellement par échange strict de leurs adresses de retour. Ce type de collaboration non-hiérarchique entre sous-programmes est connu sous le nom de

exemples: collaboration hiérarchique entre A et B

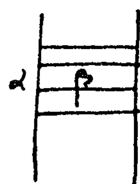


coroutines



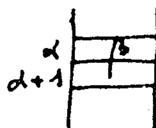
exemple : soit la pile

R8
α



et JSR 9, @(8)+
γ ...

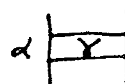
ça fait : 1°/ instruction décodée



β → aux

2°/ empiler la valeur courante de R9

R8
α



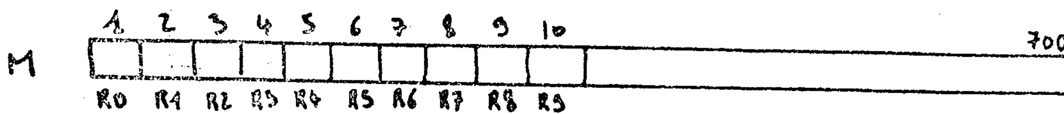
3°/ aux → R9 , i.e. β

Pour bien saisir ce mécanisme, revoyez p. 38 ce que fait exactement JSR.

DESCRIPTION DU SIMULATEUR.

La mémoire, c'est 1 tableau FORTRAN [1 : 700]

M



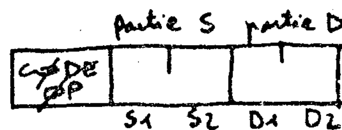
Les registres R0 à R9 occupent les 10 premiers mots du tableau. Au début de l'exécution d'un programme, le pointeur de pile R8 est initialisé à 701.

Voici comment est codée une instruction.

sur 5 chiffres 

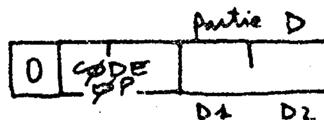
Y a 2 types d'instructions

1°/ instructions à 2 adresses



CODE OP : 1 chiffre $\in [1, 9]$

2°/ instructions à 0 et 1 adresse



CODE OP : 2 chiffres $\in [13, 30]$

registres : 1 chiffre $\in [0, 9]$.

PC $\equiv$ R9 compteur ordinal.

Voici les codes :

2 adresses	MØV	1	cc, D ← S
	CMP	2	cc ← D − S
	ADD	3	cc, D ← D + S
	SUB	4	cc, D ← D − S
	MUL	5	cc, D ← D * S
	DIV	6	cc, D ← D / S
	RI	7	lire M[S] entiers en I8 à partir de D
			→ avec M[S] ∈ [1,10]
	WI	8	écrire M[S] entiers en I8 à partir de D
	WA	9	écrire M[S] caractères en A1 à partir de D
			→ M[S] ∈ [1,80]

1 adresse	BR	13	PC ← D i.e. vers D
	BNE	14	PC ← D si cc ≠ 0
	BEQ	15	PC ← D si cc = 0
	BGT	16	PC ← D si cc > 0
	BGE	17	PC ← D si cc ≥ 0
	BLT	18	PC ← D si cc < 0
	BLE	19	PC ← D si cc ≤ 0
	BBR	20	PC ← D si une opération de lecture est en cours
	BBW	21	PC ← D si une opération d'écriture est en cours
	RTS	22	PC ← reg , PØP (reg)
	CLR	23	cc, D ← 0
	INC	24	cc, D ← D+1
	DEC	25	cc, D ← D−1
	NEG	26	cc, D ← −D
	HALT	27	STØP
	TST	28	cc ← D
	RA	29	lire 80 caractères en A1 sur une carte à partir de D
	JSR	30	PUSH (reg) ; PC → reg ; D → PC

ADRESSAGE — on a S = S1 S2

D = D1 D2 , on va les appeler des MØDES.

Soit un mode : 2 chiffres constitutifs c1 c2 .

c2 spécifie un registre ∈ [0,9]

c1 ∈ [0,7]

c1 : direct	indirect	adressage
0	1	registre
2	3	auto-incrément et immédiat (c2 est alors R9)
4	5	auto-décrément
6	7	indexé et relatif → (c2 est alors R9)

exemples :

MØV 1,2 donnera	1	01	02
MØV @1,@2	1	11	12
MØV 0,-(1)	1	00	41
MØV -(2),@(4)+	1	42	34
MØV (1)+,@-(3)	1	21	53

exemples : MØV =-1,2

1	29	02
		-1

↓ l'opérande immédiat est stocké dans le mot suivant

MØV =700,@5

1	29	15
		700

exemples : soit ADR1 ≡ 200 , ADR2 ≡ 300

MØV ADR1,0

1	69	00
		200

MØV 1,ADR2

1	01	69
		300

MØV ADR1,ADR2

1	69	69
		200
		300

↓ les 2 opérandes sont stockés dans les mots suivants.

exemples

MØV @ADR2,@ADR1

1	79	79
		300
		200

MØV ADR1(0),ADR2(0)

1	60	60
		200
		300

MØV @ADR2(3),@ADR1(5)

1	73	75
		300
		200

MØV =ADR1,@=ADR2

1	29	39
		200
		300

Donc il est clair que

MØV =C,0 est équivalent à MØV (9)+,0
DC C

MØV C,0 est équivalent à MØV @=C,0
et à MØV @(9)+,0
DC C

Donc, pour revenir à la remarque de tout à l'heure (p. 22) sur la question de savoir ce qui se produit dans le cas de

MØV =1, =2 , ça s'assemble en

1	29	29
		1
		2

et, après exécution

1	29	29
		1
		1

Étant donné un mode C1 C2, on obtiendra le décodage que voici, qui ramène dans IAD l'adresse d'opérande.

MØDULE DE DÉCØDAGE

```
                 C2 → reg ;
                     gotø (registre, autø-inc, autø-dec, indexé), C1+1
registre :      iad ← reg ; vers indirect ;
autø-inc : (et immédiat)  :  iad ← M [reg] ;
          si reg = 9            M [reg] ← M [reg] + 1 ;
                                vers indirect ;
autø-dec :   M [reg] ← M [reg] - 1 ; iad ← M [reg] ; vers indirect ;
indexé  (et relatif si reg = 9) :  iad ← si reg = 9 alors M [reg]
                                         sinon M [PC] + M [reg] ;
indirect  :  si C1 ∈ [1, 3, 5, 7]  alors  iad ← M [iad]
EXIT
```

Voici les temps associés aux instructions, en unités arbitraires qu'on suppose très petites.

MØV, CMP, ADD, SUB, BR, BNE, BEQ, BGT, BGE, BLT, BLE, BBR, BBW, CLR, INC, DEC, NEG, TST, HALT : 1U
MUL = 5U
DIV = 6U
RTS = 2U
JSR = 3U

RI, RA, WI, WA : 50 U au minimum, ça a l'air vraiment modeste, mais si j'avais mis plus vraisemblable, i.e. ≃ 500-5000 U, le simulateur aurait eu l'air de s'arrêter pour longtemps, ce qui est psychologiquement déprimant (voir aussi KNUTH, volume 1, p. 208, exercice [6]).

L'emploi de l'indirection, de l'indexation, du mode immédiat, et du mode relatif allonge l'exécution d'1 U.

exemples :		TEMPS
MØV	0,1	1U
MØV	@0,1	2U
MØV	@0,@1	3U
MØV	X(0),1	2U
MØV	X(0),Y(1)	3U
MØV	@X(0),@Y(1)	5U
MØV	=1,2	2U
MØV	2,B	2U
MØV	(0)-,+(1)	1U

PATRICK GREUSSAY.

VINCENNES

1973

LISTING DU SYSTEME QUASI-11
EN FØRTRAN IV CAE 510.
ASSEMBLEUR + SIMULATEUR.

```
C
C
      COMMON IBUF(28),IET,ICARSP(19),M(705)
C
      DIMENSION IUBOP(31),ISD(4),IETIQS(5,141)
C
C
C
      EQUIVALENCE (IDEB,IBUF(1)),(JCOD,IBUF(5)),(IZAD,IBUF(6))
      EQUIVALENCE (ISD(1),IS),(ISD(2),ID),(ISD(3),IS1),(ISD(4),ID1)
      EQUIVALENCE (IPC,M(705))
C
      EQUIVALENCE (IBOO,M(1)),(ICC2,M(2)),(I,M(3)),(J,M(4))
      EQUIVALENCE (IX,M(5)),(IY,M(6)),(IND,M(7)),(IPO,M(8))
      EQUIVALENCE (IPCXEQ,M(10))
      EQUIVALENCE (IETIQS(1),M(1))
C
C                           FORMATS
C
201   FORMAT(80A1)
202   FORMAT(4A1,2H  ,A4,3H   ,18A1,5A6)
203   FORMAT(16A4)
204   FORMAT(3X,4A1,I8)
205   FORMAT(10X,A1,9H*QUASI-11)
C
C                   READ DATA.  BOP,UOP,SPECIAL CHARS.
C
      ID=62415
C           ROUGE
      PRINT 205,ID
      CALL SSWTCH(4,ID)
      IF(ID.EQ.2) GOTO 1
C
      READ(19,203)(IUBOP(ID),ID=1,16)
      READ(19,203)(IUBOP(ID),ID=17,31)
      READ(19,201) ICARSP,(M(ID),ID=701,704)
C
C
      EQUIVALENCE (ISTAR,ICARSP(14))
      EQUIVALENCE (IBLAN,ICARSP(15)),(IARON,ICARSP(16))
      EQUIVALENCE (IVIR,ICARSP(17)),(ISLASH,ICARSP(18))
      EQUIVALENCE (LPAR,ICARSP(1)),(MOINS,ICARSP(2)),(IPLUS,ICARSP(13))
```

```
C
C                           FIRST  PASS
C
1       DO 2 ID=1,700
2       M(ID)=0
        IPC=11
        IET=141
        IBOO=1
C
C                     READ CARD.
C
300     READ(19,202)IBUF
        IF(IDEB.EQ.ISTAR) GOTO 300
        IF(IDEB.EQ.IBLAN) GOTO 304
C
C                     LABEL
C
        IF(NETIQ(1))302,302,301
C                                          ERROR.MULTIPLE LABEL.
301     CALL ERROR(1)
        GOTO 304
C
C                     STORE LABEL.
C
302     IET=IET-1
        DO 303 J=1,4
303     IETIQS(J,IET)=IBUF(J)
        IETIQS(5,IET)=IPC
C
C                     PSEUDO OR CODE
C
304     DO 305 I=28,31
        IF(JCOD-IUBOP(I)) 305,315,305
305     CONTINUE
C                                 IS CODE.SCAN ADRESS FIELD.
        IPC=IPC+1
        IPO=5
C                                 LOOP FIELDS 1 AND 2.
306     IPO=IPO+1
        J=IBUF(IPO)
        IF(J.EQ.IBLAN) GOTO 300
C                              CAS CODE SANS OPERANDE(S) EX HALT.
        IF(J.EQ.IARON) IPO=IPO+1
        J=IBUF(IPO)
        IX=IBUF(IPO+1)
        IF(J.EQ.LPAR) GOTO 310
        IF(J.NE.MOINS) GOTO 307
        IF(IX-LPAR) 309,310,309
307     DO 308 I=3,12
        IF(J-ICARSP(I)) 308,311,308
```

```
308    CONTINUE
C                                 INCREMENT PC
309    IPC=IPC+1
C                                 LOOP IN FIELD  1 OR 2.
310    IPO=IPO+1
       J=IBUF(IPO)
C                                 SEARCH FO BLANK OR  COMMA. IN FIELD
       IF(J.EQ.IBLAN) GOTO 300
       IF(J-IVIR) 310,306,310
C
311    IF(IX.EQ.IBLAN.OR.IX.EQ.IVIR) GOTO 310
       GOTO 309
C
C
C                      PSEUDO. ORG,END,RES,DC.
C
315    I=I-27
       GOTO (320,399,340,350),I
C              ORG END RES DC
C
C                      ORG
C                                 I,J DUMMIES.
320    IPO=6
       I=ITYCAR(J)
       IPC=ICC2
       GOTO 300
C                      END . TO 2D PASS.
C
C                      RES
340    IPC=IPC+ICONV(6)
       GOTO 300
C
C              DC
350    IF(IZAD.NE.ISLASH) GOTO 360
       IPO=7
351    IPC=IPC+1
       IPO=IPO+1
       IF(IBUF(IPO)-ISLASH)351,300,351
C
360    IPC=IPC+1
       GOTO 300
C
C
C
C
399    DO 12000 J=IET,141
12000  PRINT 204,(IETIQS(I,J),I=1,5)
       IPC=11
       STOP
```

```
C                                2D PASS
C
C                         READ CARD
C
400   READ(19,202) IBUF
      IF(IDEB-ISTAR)402,401,402
C
C                         PRINT COMMENT
401   CALL SSWTCH(1,J)
      IF(J.EQ.1) GOTO 400
      I=29
4010  I=I-1
      IF(IBUF(I).EQ.IBLAN) GOTO 4010
      PRINT 202,(IBUF(J),J=1,I)
      GOTO 400
C
C                         PSEUDO OR CODE
C
C                                 IX,IY INDEX DE ISD
402   IX=1
      IY=3
      IS=0
      ID=0
      IS1=(-100000)
      ID1=(-100000)
      IPO=6
C                            INIT FOR FIELD.CONVENIENT HERE FOR PRINTING.
      DO 403 I=1,31
      IF(JCOD-IUBOP(I))403,404,403
403   CONTINUE
C                                   ERROR . UNKNOWN CODE.
      CALL ERROR(2)
      GOTO 482
C                         IS CODE OR PSEUDO
404   IF(I-28)405,490,490
C
C                         IS CODE
C
C
C                         CODE FOUND. SCAN ADRESS FIELD.
405   ICOD=I
C                                   UOP=1-18,BOP=19-27.
      IF(ICOD.EQ.15) GOTO 480
C                                             HALT, WITHOUT OPERANDS.
      IF(ICOD.LT.18)IX=2
C                                   LOOP FIELDS 1 AND 2
408   IND=0
      IF(IBUF(IPO).NE.IARON)GOTO 409
C                         DEFERRED
      IND=1
      IPO=IPO+1
```

```
C
C                              GET TYPE
C
409    I=ITYCAR(ICC)
       GOTO(410,430,450,420,440),I
C           REG INC DEC IMM INDX AND RELAT
C
C                  REGISTER            R
410    J=0
       GOTO 470
C
C                  IMMEDIATE           =AD , =NB
420    ISD(IY)=ICC2
C                  AUTO INCREMENT      (R)+
430    J=2
       GOTO 470
C
C                  INDEXED AND RELATIVE AD(R),NB(R),AD,NB
440    ISD(IY)=ICC2
       J=6
       GOTO 470
C
C                  AUTO DECREMENT      -(R)
450    J=4
C
C           FORM XX AND PUT IT IN ISD(IX)
C           ISD(IY) EVENTUALLY CONTAINS AD OR NB.
C
470    ISD(IX)=(J+IND)*10+ICC
C
C            ICC=REGISTER 0-9
C            IND=PARITY , DEFERRED OR NOT 1-0
C            J  =MODE 0,2,4,6
C
       IF(IBUF(IPO).NE.IVIR) GOTO 480
       IPO=IPO+1
       IX=2
       IY=4
       GOTO 408
C
C
C            ASSEMBLE  XX IN IS,XX IN ID
C            EVENTUALLY  AD,NB IN IS1 OR/AND ID1
C
480    IF(ICOD.GT.18) GOTO 481
C                                     1-18 TO 01300-03R00
       ICOD=(ICOD+12)*100
       GOTO 482
```

```
C
C                        BOP                          19-27 TO 10000-90000
C
481   ICOD=(ICOD-18)*10000
C
482   M(IPC)=ICOD+100*IS+ID
C                                        ASSEMBLY PRINTING
C
C
483   IF(IET*5-4.LE.IPC) CALL ERROR(4)
      CALL PRINTP(2)
      IPC=IPC+1
C                         PRINT EVENTUALLY REST OF INSTRUCTION.
C
      DO 484 I=3,4
      IF(ISD(I).EQ.(-100000)) GOTO 484
      M(IPC)=ISD(I)
      CALL PRINTP(0)
      IPC=IPC+1
484   CONTINUE
      GOTO 400
C
C                                       PSEUDO. ORG,END,RES,DC
C
C
490   I=I-27
      IF(I.LT.4) CALL PRINTP(2)
      GOTO(491,492,493,494),I
C          ORG END RES DC
C                                     I,J,DUMMIES. VALUE IN ICC2.
C
C                        ORG
491   I=ITYCAR(J)
      IPC=ICC2
      GOTO 400
C
C                        RES
493   IPC=IPC+ICONV(6)
      GOTO 400
C
C                        DC
494   IF(IZAD.NE.ISLASH) GOTO 499
      M(IPC)=IBUF(7)
      CALL PRINTP(2)
      IPO=7
495   IPC=IPC+1
      IPO=IPO+1
      J=IBUF(IPO)
      IF(J.EQ.ISLASH) GOTO 400
      M(IPC)=J
      CALL PRINTP(0)
      GOTO 495
```

```
C
499   I=ITYCAR(J)
C
500   M(IPC)=ICC2
      CALL PRINTP(2)
      IPC=IPC+1
      GOTO 400
C
C
C                    END
492   I=ITYCAR(J)
      ID=ICC2
C
C
C     ********************************** RUN ****************************
C
C           TEST ASMB ERROR ELSE EXECUTE.
C
      ID1=IBOO
9999  IPCXEQ=ID
      IF(ID1.EQ.1) CALL EXEC
      STOP
      CALL SSWTCH(5,I)
      IF(I-1) 1,9999,1
      END
      A
```

```
C
      SUBROUTINE EXEC
C                            EXECUTION MODULE.
C
      COMMON IBUF(28),IET,ICARSP(19),M(705)
C
C
      LOGICAL FINOUT
      FINOUT(I)=I.NE.0.AND.I.LE.ITIMES
C
      EQUIVALENCE (KIN,IBUF(7)),(KOU,IBUF(8)),(ITIMES,IBUF(9))
      EQUIVALENCE (IS4,IBUF(10)),(ID4,IBUF(11)),(JIN,IBUF(12))
      EQUIVALENCE (JOU,IBUF(13)),(IS3,IBUF(14)),(ID3,IBUF(15))
      EQUIVALENCE (IPC,M(10)),(ISP,M(9))
      EQUIVALENCE (ID,IBUF(1)),(ICOD,IBUF(2)),(LR,IBUF(19))
      EQUIVALENCE (INS,IBUF(20)),(ID1,IBUF(21)),(IS1,IBUF(6))
      EQUIVALENCE (IZAD,IBUF(16)),(ICC,IBUF(17)),(LASTIM,IBUF(18))
C
C                    FORMATS
200   FORMAT(10I8)
201   FORMAT(80A1)
205   FORMAT(///10X,I8,7H  UNITS///)
C
      KIN=0
      KOU=0
      ITIMES=0
      LASTIM=0
      ISP=701
C
C                     RUN
C
C             NEXT  -  DECODE
C
10000 INS=M(IPC)
      ITIMES=ITIMES+LASTIM
      LASTIM=1
      CALL SSWTCH(2,ID)
      IF(ID.EQ.1) PRINT 200,(M(ID),ID=1,10)
C                                   TRACE SI CLE 2
      CALL SSWTCH(3,ID)
      IF(ID.EQ.1) RETURN
C                                   TRAP SI CLE 3
C
      IF(FINOUT(KIN)) GOTO 10005
10013 IF(FINOUT(KOU)) GOTO 10009
C
10014 IPC=IPC+1
      ICOD=INS/10000
C                                 UOP=0 , BOP=1,9
      IF(ICOD)10004,10001,10004
```

```
C
C                     UOP
C                                 0 TO 13-30 TO 1-18
10001 ICOD=INS/100-12
      INS=MOD(INS,100)
C                                 CC0XX EXCEPT RTS=0220R , JSR=03RXX
      IF(ICOD.EQ.10)LR=INS+1
C           RTS                             LR=1-10
      IF(ICOD-18)10003,10002,10002
C                                           JSR   18-27 TO ICOD=18,LR=1-10
10002 LR=ICOD-17
      ICOD=18
C
10003 ID=IDECOD(INS)
      ID1=M(ID)
      IZAD=ID+79
C                                           FOR RA
C          BR  BNE BEQ BGT BGE BLT BLE BBR BBW RTS CLR
      GOTO(800,801,802,803,804,805,806,807,808,809,810,
     *811,812 ,813,814 ,815,816,817),ICOD
C     INC DEC  NEG HALT TST RA  JSR
C
C                     BOP
C                                 CXXXX
10004 INS=MOD(INS,10000)
      IS=IDECOD(INS/100)
      ID=IDECOD(MOD(INS,100))
      IS1=M(IS)
      ID1=M(ID)
      IZAD=ID+IS1-1
C                                           FOR RI,WI,WA
      GOTO(700,701,702,703,704,705,706,707,708),ICOD
C          MOV CMP ADD SUB MUL DIV RI   WI   WA
C
C            ID = DESTINATION
C            IS = SOURCE
C
C                                 BOP
C                     MOV
700   ICC=IS1
      GOTO 900
C                     CMP
701   ICC=ID1-IS1
      GOTO 10000
C                     ADD
702   ICC=ID1+IS1
      GOTO 900
C                     SUB
703   ICC=ID1-IS1
      GOTO 900
```

```
C
C                        MUL
704    ICC=ID1*IS1
       LASTIM=5
       GOTO 900
C                        DIV
705    ICC=ID1/IS1
       LASTIM=6
       GOTO 900
C                        RI
706    JIN=1
       GOTO 709
C                        WI
707    JOU=1
       GOTO 710
C                        WA
708    JOU=2
       GOTO 710
C
C
C                        INIT I/O.
C
C            INIT INPUT.
709    IF(KIN.NE.0) GOTO 10000
       KIN=ITIMES+50
       IS3=ID
       ID3=IZAD
       GOTO 10000
C
C            INIT OUTPUT.
710    IF(KOU.NE.0) GOTO 10000
       KOU=ITIMES+50
       IS4=ID
       ID4=IZAD
       GOTO 10000
C
C                                    UOP
C
C                         BR
800    IPC=ID
       GOTO 10000
C                         BNE BEQ BGT BGE BLT BLE
801    IF(ICC)800,10000,800
802    IF(ICC)10000,800,10000
803    IF(ICC)10000,10000,800
804    IF(ICC)10000,800,800
805    IF(ICC)800,10000,10000
806    IF(ICC)800,800,10000
```

```
C
C                        BBR
807    IF(KIN) 800,10000,800
C                        BBW
808    IF(KOU) 800,10000,800
C                        RTS
809    IPC=M(LR)
       M(LR)=M(ISP)
       ISP=ISP+1
       LASTIM=2
       GOTO 10000
C                        CLR
810    ICC=0
       GOTO 900
C                        INC
811    ICC=ID1+1
       GOTO 900
C                        DEC
812    ICC=ID1-1
       GOTO 900
C                        NEG
813    ICC=(-ID1)
       GOTO 900
C                        HALT
814    PRINT 205,ITIMES
       RETURN
C                        TST
815    ICC=ID1
       GOTO 10000
C                        RA
816    JIN=2
       GOTO 709
C                        JSR
817    ISP=ISP-1
       M(ISP)=M(LR)
       M(LR)=IPC
       IPC=ID
       LASTIM=3
       GOTO 10000
C
C
900    M(ID)=ICC
       GOTO 10000
C
C                        EXECUTE READ.
C
10005 GOTO (10006,10007),JIN
C
C                  EXEC RI.
10006 READ(19,200) (M(ID),ID=IS3,ID3)
       GOTO 10008
```

```
C
C                  EXEC RA.
10007 READ(19,201) (M(ID),ID=IS3,ID3)
C
C              SET READ READY AGAIN.
10008 KIN=0
      GOTO 10013
C
C                         EXECUTE WRITE.
C
10009 GOTO (10010,10011),JOU
C
C                  EXEC WI
10010 PRINT 200,(M(ID),ID=IS4,ID4)
      GOTO 10012
C
C                  EXEC WA
10011 PRINT 201,(M(ID),ID=IS4,ID4)
C
C              SET WRITE READY AGAIN.
10012 KOU=0
      GOTO 10014
C
C
      END
      A
```

```
C
      FUNCTION NETIQ(KK)
C
C     CHERCHE SI IBUF(KK --- JUSQUE VIR,BLAN,LPAR,PLUS,MOINS,
C     OU 4 CARS EST DEJA DANS IETIQS.
C                       OUI= RAMENE ADRESSE IETIQS(I,5)
C                       NON=     ZERO.
C     DANS TOUS LES CAS, PLACE IPO SOUS CAR SUIVANT ETIQUETTE.
C     IT(1-4) = AUXILIAIRE POUR STOCKER ETIQ.
C
      COMMON IBUF(28),IET,ICARSP(19),M(705)
      COMMON /COM2/ I,J,IT(4)
      DIMENSION IETIQS(5,141)
C
      EQUIVALENCE (IPLUS,ICARSP(2)),(MOINS,ICARSP(13))
      EQUIVALENCE (IBLAN,ICARSP(15)),(IVIR,ICARSP(17))
      EQUIVALENCE (LPAR,ICARSP(1)),(IPO,M(8))
      EQUIVALENCE (IETIQS(1),M(1))
C
C
      IPO=KK
      NETIQ=0
      DO 1  I=1,4
1     IT(I)=IBLAN
      DO 2  I=1,4
      IT(I)=IBUF(IPO)
      IPO=IPO+1
      J=IBUF(IPO)
      IF(J.EQ.IVIR.OR.J.EQ.LPAR.OR.J.EQ.IBLAN.OR.
     XJ.EQ.IPLUS.OR.J.EQ.MOINS) GOTO 3
2     CONTINUE
3     DO 5  I=IET,141
      DO 4  J=1,4
      IF(IT(J)-IETIQS(J,I))5,4,5
4     CONTINUE
      NETIQ=IETIQS(5,I)
      RETURN
5     CONTINUE
      RETURN
      END
      A
```

```
C
      FUNCTION ICONV(IPO1)
C
C     CONVERTIT A PARTIR DE IPO1
C     ET POINTE IPO SOUS CAR SUIVANT LE NB.
C
      COMMON IBUF(28),IET,ICARSP(19),M(705)
      COMMON /COM2/ ISIG,K,I
      EQUIVALENCE (MOINS,ICARSP(2)),(IPLUS,ICARSP(13))
      EQUIVALENCE (IPO,M(8))
C
C
      IPO=IPO1
      I=IBUF(IPO)
      ISIG=1
      K=0
      IF(I.NE.MOINS) GOTO 1
      ISIG=(-1)
      GOTO 2
1     IF(I-IPLUS)3,2,3
2     IPO=IPO+1
3     DO 4 I=3,12
      IF(IBUF(IPO)-ICARSP(I))4,6,4
4     CONTINUE
5     ICONV=K*ISIG
      RETURN
6     K=K*10+I-3
      GOTO 2
      END
      A


C
      FUNCTION MOD(I,J)
      MOD=I-(I/J)*J
      RETURN
      END
      A
```

```
C
      SUBROUTINE ERROR(I)
      COMMON IBUF(28),IET,ICARSP(19),M(705)
      COMMON /COM2/ KK
      EQUIVALENCE (IBOO,M(1))
C
      IBOO=62415
C           ROUGE
      PRINT 100,IBOO,I
      KK=4
      IF(I.EQ.1) KK=3
      CALL PRINTP (KK)
C             POUR IMPRESSION LIGNE ERREUR.
      IBOO=0
      RETURN
C
100   FORMAT(A1,4H*ER ,I8)
      END
      A
```

```
C
      FUNCTION ITYCAR(ICC)
C
C             CALLED BY 409 PP.
C     DETERMINE LE TYPE DU FIELD COURANT,
C     PUIS POINTE IPO SOUS LE CAR SUIVANT LE FIELD.
C     RAMENE NO TYPE 1-5 , ICC , ET ICC2 SI YA LIEU.
C
C                                NO TYPE    ICC         ICC2
C              R                  1          0-9         NIL
C              (R)+               2          0-9         NIL
C              -(R)               3          0-9         NIL
C              =AD     =NB        4          9           AD OU NB
C              AD(R)  NB(R)       5          0-9         AD OU NB
C              AD      NB         5          9           AD OU NB
C
      COMMON IBUF(28),IET,ICARSP(19),M(705)
      COMMON /COM2/ KK
C
      EQUIVALENCE (LPAR,ICARSP(1)),(MOINS,ICARSP(2)),(ISTAR,ICARSP(14))
      EQUIVALENCE (IPC,M(705)),(K,ICC2,M(2)),(IX,M(9))
      EQUIVALENCE (IEGAL,ICARSP(19)),(IPO2,IPO,M(8))
      EQUIVALENCE (I,M(3)),(J,M(4))
      EQUIVALENCE (IBLAN,ICARSP(15)),(IVIR,ICARSP (17))
C
      J=IBUF(IPO)
      KK=IBUF(IPO+1)
C                      IS REG               R
      DO 1 I=3,12
      IF(J-ICARSP(I))1,7,1
1     CONTINUE
C                      IS AUTO INC          (R)+
      IF(J.EQ.LPAR) GOTO 8
C                      IS AUTO DEC          -(R)
      IF(J.EQ.MOINS.AND.KK.EQ.LPAR) GOTO 9
C                      IS IMMEDIATE         =AD =NB
      ICC=9
      IF(J.EQ.IEGAL) GOTO 2
C
C                      IS RELATIVE OR INDEXED
14    IX=2
      GOTO 3
C
C             IMMEDIATE    .    IX USED FOR RETURN IF NOT INDEXED.
2     ITYCAR=4
      IX=1
      IPO=IPO+1
3     K=0
C
C             COMMON PART FOR IMMEDIATE,RELATIVE,INDEXED
C             GET ADRESS OR NUMBER
```

```
C
C                                   ADRESS .+NB,.-NB
12     DO 4 I=2,13
       IF(IBUF(IPO)-ICARSP(I))4,6,4
4      CONTINUE
C                                             ADRESS GET LABEL.
       K=NETIQ(IPO2)
       IF(K)6,5,6
C                                             ERROR ADRESS UNKNOWN
5      CALL ERROR(3)
       RETURN
C                                             NUMBER . CONVERT IT.
6      ICC2=K+ICONV(IPO2)
C                                             K=IPC OR 0
C
11     IF(IX.EQ.1) RETURN
C                                             IX=1  IMMEDIATE
C                    RELATIVE OR INDEXED
       ITYCAR=5
       IF(IBUF(IPO).NE.LPAR) RETURN
C                                   RETURN IF RELATIVE
C                    INDEXED                  AD(R) NB(R)
       ICC=ICONV(IPO2+1)
       GOTO 10
C
C                    REGISTER                 R
7      IF(KK.NE.IBLAN.AND.KK.NE.IVIR) GOTO 14
       ITYCAR=1
       ICC=I-3
       GOTO 10
C
C                    AUTO INCREMENT           (R)+
8      ITYCAR=2
       ICC=ICONV(IPO2+1)
       IPO=IPO+2
       RETURN
C
C                    AUTO DECREMENT           -(R)
9      ITYCAR=3
       ICC=ICONV(IPO2+2)
10     IPO=IPO+1
       RETURN
       END
       A
```

```
C
      FUNCTION IDECOD(IXX)
C
C           ADRESSAGE PDP 11.         10 REGISTRES.
C           RAMENE DANS IS OU ID L ADRESSE D OPERANDE.
C
C           IXX == XR     R=0-9   XX/2=0,2,4,6
C                                  SI INDIRECT 1,3,5,7
C           0      REGISTER
C           2      AUTO INC , IMMEDIAT SI R=9
C           4      AUTO DEC
C           6      INDEXED , RELATIVE SI R=9
C
      COMMON IBUF(28),IET,ICARSP(19),M(705)
C
      EQUIVALENCE (IX1,IBUF(3)),(IAD,IBUF(4)),(IREG,IBUF(5))
      EQUIVALENCE (IPC,M(10))
      EQUIVALENCE (LASTIM,IBUF(18))
C
      IX1=IXX/10
      IAD=IX1/2+1
      IREG=MOD(IXX,10)+1
      GOTO(100,101,102,103),IAD
C          REG INC DEC INDX
C
C                   REG
100   IAD=IREG
      GOTO 105
C                   AUTO INC   ET   IMMEDIAT
101   IF(IREG.EQ.10) GOTO 1011
      IAD=M(IREG)
      M(IREG)=M(IREG)+1
      GOTO 105
C                   IMMEDIAT
1011  IAD=IPC
      GOTO 104
C                   AUTO DEC
102   M(IREG)=M(IREG)-1
      IAD=M(IREG)
      GOTO 105
C                   INDEXED ET RELATIVE
103   IF(IREG.EQ.10) GOTO 1031
      IAD=M(IPC)+M(IREG)
      GOTO 104
C                   RELATIVE
1031  IAD=M(IPC)
104   LASTIM=LASTIM+1
      IPC=IPC+1
C
C              INDIRECTION SI IX1 IMPAIR
105   IF(MOD(IX1,2))106,107,106
106   IAD=M(IAD)
      LASTIM=LASTIM+1
107   IDECOD=IAD
      RETURN
      END
      A
```

```
C
      SUBROUTINE PRINTP(I)
C
C     SI J = 1                YA LA CLE
C             I=0 RETURN
C             I=2 RETURN
C             I=4 LISTER IPC,M(IPC),BUFFER      (FROM ERROR)
C     SI J = 2                YA PAS LA CLE
C             I=0 LISTER IPC,M(IPC)
C             I=2 LISTER IPC,M(IPC),BUFFER
C             I=4 RETURN                        (FROM ERROR)
C
      COMMON IBUF(28),IET,ICARSP(19),M(705)
      EQUIVALENCE (IPC,M(705)),(J,M(7))
      EQUIVALENCE (IBLAN,ICARSP(15))
      COMMON /COM2/ K,L
C
      CALL SSWTCH(1,J)
      K=I+J
      GOTO (4,1,4,2,2,4),K
1     PRINT 200,IPC,M(IPC)
      RETURN
2     K=29
3     K=K-1
      IF(IBUF(K).EQ.IBLAN) GOTO 3
      PRINT 200,IPC,M(IPC),(IBUF(L),L=1,K)
4     RETURN
C
200   FORMAT(2I8,4X,4A1,2X,A4,2X,18A1,5A6)
      END
```

```
BR  BNE BEQ BGT BGE BLT BLE BBR BBW RTS CLR INC DEC NEG HALTTST
RA  JSR MOV CMP ADD SUB MUL DIV RI  WI  WA  ORIGEND RES DC
(-0123456789+* ¢,/=.
```

```
*
*       INVERSION D UN ENTIER POSITIF.
*
BLAN    DC      / /
E1      CLR     1
        BBW     .
        BR      E3
*
E2      MOV     0,-(8)
        DIV     =10,0
        SUB     0,1
        MUL     =10,1
        ADD     (8)+,1
E3      TST     0
        BNE     E2
        WI      =1,1
        BBW     .
        WA      =1,BLAN
READ    RI      =1,0
        BBR     .
        WI      =1,0
        TST     0
        BNE     E1
        BBW     .
        HALT
*
        END     READ
```

E1. L'entier à inverser est dans N.
E2. K ← 0.
E3. Si N = 0, l'entier inversé est dans K. Fini.
E4. X ← N. Ici, N est simplement empilé.
E5. N ← N/10.
E6. K ← (K−N)∗10 + X. Vers E3.

Pour l'emploi de la pile standard, voir page 36.
R0 ≡ N, R1 ≡ K.

5
5

1234
4321

1551
1551

572
275

0

965 UNITS

```
*
*       FACTORIELLES RECURSIVES.
*
DEBU    CLR     1
RE      MOV     1,0
        JSR     9,FACT
        BBW     .
        MOV     1,N
        MOV     0,N+1
        WI      =2,N
        INC     1
        CMP     =9,1
        BNE     RE
        BBW     .
        HALT
*
FACT    TST     0
        BNE     SUIT
        MOV     =1,0
        RTS     9
SUIT    MOV     0,-(8)
        DEC     0
        JSR     9,FACT
        MUL     (8)+,0
        RTS     9
*
N       RES     2
*
        END     DEBU
```

FACT . Si M=0, M←1 . Retour.
F2 . Empiler M .
F3 . M←M−1 .
F4 . appeler FACT .
F5 . M←M * sommet de pile . Dépiler.
F6 . Retour .

R0 ≡ M

Pour l'emploi de JSR avec R9 comme registre de liaison, voir page 38.

0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320

828 UNITS

```
*
*       HANOI
*
        ORIG    30
N       RES     1
BUF     DC      /DISQUE 0 DE 0 A 0/
NUM     DC      /123456789/
*
HAN1    MOV     1,-(8)
        MOV     2,-(8)
        MOV     3,-(8)
        MOV     4,-(8)
        DEC     1
        MOV     @8,3
        MOV     1(8),4
        JSR     9,HANO
        MOV     (8)+,2
        MOV     (8)+,3
        MOV     (8)+,4
        MOV     (8)+,1
        BBW     .
        MOV     NUM-1(1),BUF+7
        MOV     4,BUF+12
        MOV     3,BUF+16
        WA      =17,BUF
        DEC     1
HANO    TST     1
        BNE     HAN1
        RTS     9
*
GO      MOV     NUM,2
        MOV     NUM+1,3
        MOV     NUM+2,4
        MOV     N,1
        JSR     9,HANO
        BBW     .
        WA      =1,BUF+6
*
DEB     RI      =1,N
        BBR     .
        WI      =1,N
        TST     N
        BNE     GO
        BBW     .
        HALT
        END     DEB
```

m : nombre total de disques empilés.
d : n° aiguille de départ, ici 1
a : n° aiguille d'arrivée, ici 2
i : n° aiguille intermédiaire, ici 3

```
          1
DISQUE 1 DE 1 A 2

          2
DISQUE 1 DE 1 A 3
DISQUE 2 DE 1 A 2
DISQUE 1 DE 3 A 2

          3
DISQUE 1 DE 1 A 2
DISQUE 2 DE 1 A 3
DISQUE 1 DE 2 A 3
DISQUE 3 DE 1 A 2
DISQUE 1 DE 3 A 1
DISQUE 2 DE 3 A 2
DISQUE 1 DE 1 A 2

          4
DISQUE 1 DE 1 A 3
DISQUE 2 DE 1 A 2
DISQUE 1 DE 3 A 2
DISQUE 3 DE 1 A 3
DISQUE 1 DE 2 A 1
DISQUE 2 DE 2 A 3
DISQUE 1 DE 1 A 3
DISQUE 4 DE 1 A 2
DISQUE 1 DE 3 A 2
DISQUE 2 DE 3 A 1
DISQUE 1 DE 2 A 1
DISQUE 3 DE 3 A 2
DISQUE 1 DE 1 A 3
DISQUE 2 DE 1 A 2
DISQUE 1 DE 3 A 2

          0


                    2164  UNITS
```

HANØI . si $m = 0$, retour.
HAN1 . empiler m.
HAN2 . empiler d.
HAN3 . empiler a.
HAN4 . empiler i.
HAN5 . $m \leftarrow m - 1$.
HAN6 . Échanger a et i. L'instruction MØV @8,3 met dans R3 le sommet de pile, i.e. i, et MØV 1(8),4 met dans R4 le sous-sommet de pile, i.e. a. Ce qui réalise la permutation $\begin{pmatrix} d & a & i \\ d & i & a \end{pmatrix}$

HAN7 . appeler HANØI.
HAN8 . $d \leftarrow$ sommet de pile. Dépiler.
HAN9 . $a \leftarrow$ sommet de pile. Dépiler.
HAN10 . $i \leftarrow$ sommet de pile. Dépiler.
HAN11 . $m \leftarrow$ sommet de pile. Dépiler.
Ce qui réalise la permutation $\begin{pmatrix} d & a & i \\ i & a & d \end{pmatrix}$

HAN12 . Imprimer "DISQUE m DE i VERS a". N'oubliez pas que i contient l'ancienne valeur de d.

HAN13 . $m \leftarrow m - 1$. Vers HANØI.

$R1 \equiv m$, $R2 \equiv d$, $R3 \equiv a$, $R4 \equiv i$.

```
*
*       ELIMINATION DES BLANCS D UNE LIGNE.
*
        ORIG    500
POIN    DC      /./
BLAN    DC      / /
TEX     RES     80
E0      MOV     =-1,0
E1      INC     0
        CMP     TEX(0),BLAN
        BNE     E1
        MOV     0,1
        BBW     .
E2      INC     1
        CMP     TEX(1),BLAN
        BEQ     E2
        MOV     TEX(1),TEX(0)
        INC     0
        CMP     TEX(1),POIN
        BNE     E2
        WA      0,TEX
        BBW     .
        WA      =1,BLAN
        BBW     .
LIRE    RA      TEX
        BBR     .
        WA      =80,TEX
        CMP     TEX,POIN
        BNE     E0
        BBW     .
        HALT
        END     LIRE
```

E1. Placer dans I l'adresse du 1er blanc de la ligne.

E2. Placer dans J l'adresse du 1er caractère non-blanc suivant celui à l'adresse I.

E3. Ligne [J] → Ligne [I]. I ← I+1.

E4. Si Ligne [J] = ".", fini (et I contient le nombre de caractères non-blancs). Sinon vers E2.

$R0 \equiv I$, $R1 \equiv J$.

LA CIGALE AYANT CHANTE TOUT L ETE SE   TROUVA .
LACIGALEAYANTCHANTETOUTLETESETROUVA.

FORT DEPOURVUE QUAND LA BISE FUT VENUE.
FORTDEPOURVUEQUANDLABISEFUTVENUE.

1655 UNITS

```
*
*       TRIANGLE DE PASCAL.
*
E2      MOV     0,3
        INC     3
        SUB     2,3
        MUL     1,3
        DIV     2,3
        MOV     3,1
E3      MOV     1,(4)+
        INC     2
        CMP     0,2
        BLE     E2
        WI      2,LIGN
        INC     0
        CMP     =10,0
        BLT     E1
        BBW     .
        HALT
DEBU    CLR     0
E1      MOV     =1,1
        CLR     2
        MOV     =LIGN,4
        BBW     .
        BR      E3
LIGN    RES     0
        END     DEBU
```

L'algorithme est fondé sur les relations :

$\binom{n}{0} = 1$, $\binom{n}{p} = \binom{n}{p-1} * (n-p+1)/p$.

E0 . $n \leftarrow 0$.

E1 . $p1 \leftarrow 1$. $p \leftarrow 0$. Préparer le pointeur de stockage . Vers E3 .

E2 . $x \leftarrow p1 * (n-p+1)/p$. $p1 \leftarrow x$.

E3 . stocker p1 . $p \leftarrow p+1$. Si $p \leq n$ vers E2 .

E4 . écrire les p nombres stockés . $n \leftarrow n+1$.
si $n < 10$ vers E1 sinon fini .

R0 $\equiv n$, R1 $\equiv p1$, R2 $\equiv p$, R3 $\equiv x$, R4 $\equiv$ pointeur de stockage .

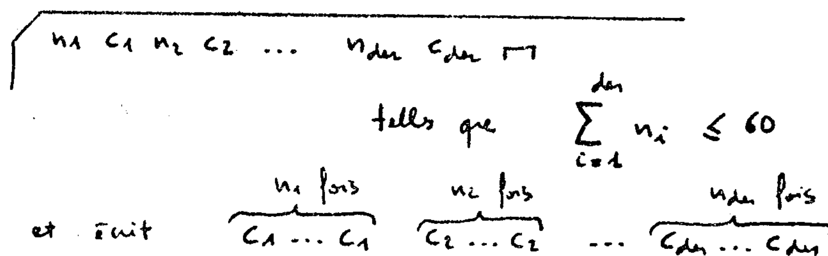
1									
1	1								
1	2	1							
1	3	3	1						
1	4	6	4	1					
1	5	10	10	5	1				
1	6	15	20	15	6	1			
1	7	21	35	35	21	7	1		
1	8	28	56	70	56	28	8	1	
1	9	36	84	126	126	84	36	9	1

1499 UNITS

```
*
*       COROUTINES.
*
        ORIG    40
POIN    DC      /./
IN      RES     80
OUT     RES     60
BLAN    DC      / /
*
ISNU    MOV     =10,6
ISN2    DEC     6
        CMP     IN(0),NUM(6)
        BNE     ISN2
        RTS     9
NUM     DC      /0123456789/
*
E0      JSR     9,@(8)+         RESUME NEXC
PP      INC     1
        MOV     IN(0),(2)+
        BR      E0
*
DEBU    MOV     =PP,-(8)        PREPARE COROUTINES.
        BR      E1
END     WA      1,OUT
        BBW     .
E1      WA      =1,BLAN
        MOV     =-1,0           R0 INDEX IN.
        CLR     1               R1 INDEX OUT.
        MOV     =OUT,2
        RA      IN
        BBR     .
        WA      =80,IN
        CMP     IN,POIN
        BNE     NEXC
        BBW     .
        HALT
NEXC    INC     0               ATTRAPE LE CARACTERE
        CMP     IN(0),BLAN      IN SUIVANT.
        BEQ     END
        JSR     9,ISNU
        INC     0
E2      TST     6               R6 NB DE CARS A ECRIRE
        BEQ     NEXC
        JSR     9,@(8)+         RESUME PP
        DEC     6
        BR      E2
*
        END     DEBU
```

Voir KNUTH, vol. I, pages 190-196.

le programme lit des cartes de type (avec n_i : chiffre $\in [0,9]$
c : caractère quelconque)



telles que $\sum_{i=1}^{der} n_i \leq 60$

et écrit $\underbrace{c_1 \dots c_1}_{n_1 \text{ fois}}\ \underbrace{c_2 \dots c_2}_{n_2 \text{ fois}}\ \dots\ \underbrace{c_{der} \dots c_{der}}_{n_{der} \text{ fois}}$

1A2B3C2A
ABBCCCAA

1C0A2B
CBB

5A4B3C2D1E
AAAAABBBBCCCDDE

9 9*4 1B1R1A1V104 9*
********* BRAVO *********

3040 UNITS

Sont mises en jeu 2 coroutines

1°/ PP stocke le caractère transmis par NEXC et redemande un caractère à NEXC.
2°/ NEXC analyse le nombre n_i de caractères à stocker dans le buffer de sortie, et passe n_i fois le caractère à écrire à PP.

Les appels de coroutines (RESUME) ont lieu par l'instruction :
JSR 9, @(8)+ (voir pages 41-42).

R1 ≡ nbc : nombre de caractères stockés, R0 ≡ inp : index du buffer d'entrée,
R6 ≡ cnb : nombre d'occurrences du caractère c_i à stocker. cnb $\in [0,9]$

PP. nbc ← nbc + 1. Stocker IN [inp]. Resume NEXC. Vers PP.

NEXC. inp ← inp + 1. Si IN [inp] = blanc, écrire les nbc caractères stockés, Fini. cnb ← traduction en entier du caractère dans IN [inp].
EZ. si cnb = 0, vers NEXC. Resume PP. cnb ← cnb − 1. Vers EZ.

```
*
*       COMPTER LES VOYELLES D UN TEXTE.
*
TEXT    RES     80
VOY     DC      /AEIOU./
NB      RES     1
SCAN    MOV     =-80,1
COMP    MOV     TEXT+80(1),3
        CMP     3,VOY+5
        BNE     SCA2
        WI      =1,NB
        BBW     .
DEB     CLR     NB
        BR      LIRE
SCA2    MOV     =VOY,0
        MOV     =-6,2
SCA3    INC     2
        BEQ     AVAN
        CMP     (0)+,3
        BNE     SCA3
        INC     NB
AVAN    INC     1
        BNE     COMP
LIRE    RA      TEXT
        BBR     .
        WA      =80,TEXT
        CMP     TEXT,VOY+5
        BNE     SCAN
        BBW     .
        HALT
        END     DEB
```

NOUS SOMMES ICI PAR LA VOLONTE DU PEUPLE, ET NOUS N EN SORTIRONS
QUE PAR LA FORCE DES BAIONETTES.
34
DE L AUDACE, ENCORE DE L AUDACE ET TOUJOURS DE L AUDACE.
23
MADAME VETO AVAIT PROMIS DE FAIRE EGORGER TOUT PARIS.
21
MAIS SON COUP A MANQUE, GRACE A NOS CANONNIERS.
17
.

11405 UNITS

```
*
*              ADDITION DE MATRICES 3*3   C=A+B
*
       ORIG    100
MA     RES     9
MB     RES     9
MC     RES     9
*
BLAN   DC      / /
RES    DC      /RESULTAT/
*
DEBU   WA      =1,BLAN
       BBW     .
       MOV     =3,0
       JSR     7,LIRM
       DC      MA
       JSR     7,LIRM
       DC      MB
       BBW     .
       WA      =8,RES
*
*        ADDITION
*
       MOV     =8,1
RE     MOV     MA(1),MC(1)
       ADD     MB(1),MC(1)
       DEC     1
       BGE     RE
*
*         ECRIRE C.
*
       CLR     1
       BBW     .
       WI      0,MC(1)          1H
       ADD     0,1
       CMP     =9,1
       BLT     .-7              VERS 1B
       BBW     .
       HALT
*
LIRM   MOV     (7)+,2
       MOV     0,1
       RI      0,@2             1H
       BBR     .
       BBW     .
       WI      0,@2
       ADD     0,2
       ADD     0,1
       CMP     =9,1
       BLE     .-10             VERS 1B
       BBW     .
       WA      =1,BLAN
       RTS     7
*
       END     DEBU
```

1	2	3
4	5	6
7	8	9
1	2	3
4	5	6
7	8	9

RESULTAT

2	4	6
8	10	12
14	16	18

831 UNITS

On notera le passage de paramètres à des sous-programmes dans les appels

```
JSR    7,LIRM        et    JSR    7,LIRM
DC     MA                  DC     MB
```

Revoir à ce propos la page 39.

```
*
*       PERMUTATIONS EN ORDRE LEXICOGRAPHIQUE.
*
        ORIG    100
GO      MOV     N,7
        MOV     7,6
        INC     6
        MOV     7,0
*
E0      CLR     NFAC(0)
        DEC     0
        BGT     E0
*
ENC     BBW     .
        CLR     PERM(7)
        MOV     7,0
        MOV     =NFAC+1,2
E1      DEC     0
        MOV     (2)+,PERM(0)
        MOV     0,1
E2      INC     1
        CMP     PERM(0),PERM(1)
        BLT     E3
        INC     PERM(1)
E3      CMP     N,1
        BLT     E2
        TST     0
        BGT     E1
*
        WI      6,PERM
        MOV     =1,0
E5      CMP     0,NFAC(0)
        BEQ     E7
        INC     NFAC(0)
        MOV     0,1
        DEC     1
E6      CLR     NFAC(1)
        DEC     1
        BGT     E6
        BR      ENC
E7      INC     0
        CMP     N,0
        BLE     E5
        BBW     .
        WA      =1,BLAN
*
READ    RI      =1,N
        BBR     .
        WI      =1,N
        TST     N
        BNE     GO
        BBW     .
        HALT
*
BLAN    DC      / /
N       RES     1
NFAC    RES     10
PERM    RES     10
*
        END     READ
```

L'algorithme consiste à engendrer tous les nombres factoriels

$$\sum_{i=1}^{n} S_i * i! < (n+1)!$$

Puis, à chacun de ces nombres factoriels, faire correspondre la permutation associée par la méthode de LEHMER.

Comme on montre facilement que MAX $(S_n, S_{n-1}, \dots S_1)$ tel que $\sum_{1}^{n} S_i * i! < (n+1)!$, est $(n, n-1, \dots, 1)$, les $S_n, S_{n-1}, \dots, S_1$ varieront de $(0, 0, \dots, 0)$ à $(n, n-1, \dots, 2, 1)$

Le vecteur NFAC contiendra les S_i, et le vecteur PERM contiendra les permutations associées.

$i \leftarrow n$.

E0. NFAC[i] ← 0, i ← i-1, si i ≥ 1 vers E0.

ENC. PERM[n] ← 0. i ← n-1.

E1. PERM[i] ← NFAC[n-i], j ← i+1.

E2. si PERM[j] < PERM[i] vers E3, sinon PERM[j] ← PERM[j]+1.

E3. j ← j+1. Si j ≤ n, vers E2. Sinon i ← i-1. Si i ≥ 0 vers E1.

E4. Imprimer n+1 nombres dans PERM. i ← 1.

E5. Si NFAC[i] = i vers E7. Sinon NFAC[i] ← NFAC[i]+1, j ← i-1.

E6. NFAC[j] ← 0, j ← j-1. Si j ≥ 1 vers E6 sinon vers ENC.

E7. i ← i+1. Si i ≤ n vers E5 sinon fini.

R0 ≡ i, R1 ≡ j, R2 ≡ pointeur de PERM,
R6 ≡ n+1, R7 ≡ n.

1			
0	1		
1	0		
2			
0	1	2	
0	2	1	
1	0	2	
1	2	0	
2	0	1	
2	1	0	
3			
0	1	2	3
0	1	3	2
0	2	1	3
0	2	3	1
0	3	1	2
0	3	2	1
1	0	2	3
1	0	3	2
1	2	0	3
1	2	3	0
1	3	0	2
1	3	2	0
2	0	1	3
2	0	3	1
2	1	0	3
2	1	3	0
2	3	0	1
2	3	1	0
3	0	1	2
3	0	2	1
3	1	0	2
3	1	2	0
3	2	0	1
3	2	1	0
0			

4680 UNITS

```
*
*        CARRES MAGIQUES
*
         ORIG     40
GO       MOV      N,0
         MOV      0,8
         DEC      0
         DIV      =2,0
*
         MOV      =1,2
RE0      MOV      8,3
         MOV      =BUF,1
         ADD      8,1
*
RE1      MOV      3,6
         ADD      0,6
         SUB      2,6
         MOV      3,7
         ADD      3,7
         SUB      2,7
         CMP      8,6
         BLT      E2
         SUB      8,6
         BR       E3
E2       TST      6
         BGE      E3
         ADD      8,6
E3       CMP      8,7
         BLE      E4
         SUB      8,7
         BR       E5
E4       TST      7
         BGT      E5
         ADD      8,7
E5       BBW      .
         MUL      8,6
         ADD      7,6
         MOV      6,-(1)
*
         DEC      3
         BNE      RE1
         WI       8,BUF
         INC      2
         CMP      8,2
         BLE      RE0
         BBW      .
         WA       =1,BLAN
*
DEB      RI       =1,N
         BBR      .
         WI       =1,N
         TST      N
         BNE      GO
         BBW      .
         HALT
BLAN     DC       / /
N        RES      1
BUF      RES      0
         END      DEB
```

L'algorithme n'est pas vraiment très récent. Il s'applique aux carrés d'ordre impair. Il est mentionné pour la 1re fois en Occident dans la "Relation du Royaume de Siam" de l'ambassadeur De la Loubère, en 1693 (Je vous recommande à cette occasion l'excellent livre de Edward Falkener "Games ancient and oriental and how to play them" réédité chez Dover et vendu pour la somme modique de $2.75. C'est plein de jeux épatants égyptiens, chinois, mongols, indiens et latins).

Étant donné 2 entiers $i, j \in [1, n]$ représentant les coordonnées en ligne (i) et colonne (j) du tableau BUF, on trouve ainsi l'objet à placer :

$b \leftarrow j - i + (n-1)/2$.

$c \leftarrow j*2 - i$.

E1. si $b \geq n$, $b \leftarrow b - n$ et vers E3.

E2. si $b < 0$, $b \leftarrow b + n$.

E3. si $c > n$, $c \leftarrow c - n$ et vers E5.

E4. si $c \leq 0$, $c \leftarrow c + n$.

E5. $BUF[i, j] \leftarrow b*n + c$.

$R6 \equiv b$, $R7 \equiv c$, $R2 \equiv (n-1)/2$,

$R2 \equiv i$, $R3 \equiv j$.

3		
4	9	2
3	5	7
8	1	6

5				
11	18	25	2	9
10	12	19	21	3
4	6	13	20	22
23	5	7	14	16
17	24	1	8	15

7						
22	31	40	49	2	11	20
21	23	32	41	43	3	12
13	15	24	33	42	44	4
5	14	16	25	34	36	45
46	6	8	17	26	35	37
38	47	7	9	18	27	29
30	39	48	1	10	19	28

9								
37	48	59	70	81	2	13	24	35
36	38	49	60	71	73	3	14	25
26	28	39	50	61	72	74	4	15
16	27	29	40	51	62	64	75	5
6	17	19	30	41	52	63	65	76
77	7	18	20	31	42	53	55	66
67	78	8	10	21	32	43	54	56
57	68	79	9	11	22	33	44	46
47	58	69	80	1	12	23	34	45

0

7796 UNITS